NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

ENHANCED PRECISION GEOLOCATION IN 4G WIRELESS NETWORKS

by

Jason Q. McClintic

March 2013

Thesis Co-Advisors: Murali Tummula
John McEachen

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** March 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Enhanced Precision Geolocation in 4G Wireless Networks			**5. FUNDING NUMBERS**
6. AUTHOR(S): Jason Q. McClintic			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: NA			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE** A
13. ABSTRACT (maximum 200 words) The objective of this thesis is to improve the performance of geolocation schema though estimating the speed of light via the refractive index of air, estimating the target velocity, and exercising receiver choice. A method for incorporating the speed of light into geolocation models is proposed in this thesis. A generic receiver choice algorithm is proposed with application to time-of-arrival, time-difference-of-arrival, and Doppler velocity estimation schemes. An object-oriented MATLAB package was developed to describe the environment, network, target behavior, simulate data, and conduct simulation study. Simulation results show that using an incorrect estimate of propagation velocity, when timing information is sufficiently precise, can yield position estimates that are, on average, significantly less accurate and less precise. Further, simulation results show that inclusion of choice enables large improvements in both the average error and the dispersion of the errors.			
14. SUBJECT TERMS geolocation, algorithms, 4G mobile communication, optimization, WiMAX, Wireless networks			**15. NUMBER OF PAGES** 251
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ENHANCED PRECISION GEOLOCATION IN 4G WIRELESS NETWORKS

Jason Q. McClintic
Lieutenant, United States Navy
B.A., University of Saint Thomas, 2008

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL
March 2013**

Author:	Jason Q. McClintic
Approved By:	Murali Tummala Thesis Advisor
	John McEachen Thesis Co-Advisor
	Clark Robertson Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The objective of this thesis is to improve the performance of geolocation schema though estimating the speed of light via the refractive index of air, estimating the target velocity, and exercising receiver choice. A method for incorporating the speed of light into geolocation models is proposed in this thesis. A generic receiver choice algorithm is proposed with application to time-of-arrival, time-difference-of-arrival, and Doppler velocity estimation schemes. An object-oriented MATLAB package was developed to describe the environment, network, target behavior, simulate data, and conduct simulation study. Simulation results show that using an incorrect estimate of propagation velocity, when timing information is sufficiently precise, can yield position estimates that are, on average, significantly less accurate and less precise. Further, simulation results show that inclusion of choice enables large improvements in both the average error and the dispersion of the errors.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

BS	Base station
FCC	United States Federal Communications Commission
DAMA	Demand Assigned Multiple Access
IEEE	Institute for Electrical and Electronics Engineering
IEEE 802.16	IEEE Standard 802.16 for Local and metropolitan area networks– Part 16: Air Interface for Broadband Wireless Access Systems
DL	Downlink
DL-MAP	Downlink Map
DVE	Doppler Velocity Estimation
E911	Enhanced 911
FA	Frequency Adjust
FDD	Frequency Division Duplex
GPS	Global Positioning System
LOS	Line of Sight
MAC	Medium Access Control Layer
MS	Mobile Subscriber
NLOS	Non-Line of Sight
OFDM	Orthogonal Frequency-Division Multiplexing

OFDMA	Orthogonal Frequency-Division Multiple Access
PHY	Physical Layer
PPM	Parts Per Million
PS	Physical Slot
RCVR	Reciever
RF	Radio Frequency
RNG_REQ	Range Request message
RNG_RSP	Range Response message
SS	Subscriber Station
TA	Timing Adjust
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TDOA	Time-Difference-of-Arrival
TOA	Time-of-Arrival
UL	Uplink
UL-MAP	Uplink Map

EXECUTIVE SUMMARY

The ability to locate a cellular handset is of growing importance in the public and private sector for provision of location-based services. Existing methods commonly assume that the speed of light is a known constant and employ the available information in a predetermined way. The 4G wireless network taken as an example in this thesis conforms to the requirements of Institute for Electrical and Electronic Engineering Standard 802.16 for local and metropolitan area networks–Part 16: Air Interface for Broadband Wireless Access Systems, especially those portions describing an orthogonal frequency-division multiple access (OFDMA) network.

The objective of this thesis is to improve the performance of geolocation schema. The methods considered are time-of-arrival (TOA), time-difference-of-arrival (TDOA), and differential Doppler (DD). Improvement is accomplished by estimating the speed of light and exercising receiver choice.

A generic receiver choice algorithm and applications to three different geolocation algorithms are proposed. The proposed method uses a simple, linear algebra-based decision rule to choose constraint equations that together may be expected to be better conditioned than the naive choice made in the original algorithms taken from the literature. This work is believed to be new to the geolocation literature. A new MATLAB package, **Geolocation**, was developed to implement the various models, algorithms, and other tools necessary to the simulation process. The computational complexity of the proposed constraint choice scheme for TOA and DD is on the order of the square of the number of receivers. For TDOA, it is on the order of the number of receivers to the fourth power.

Simulation results show addition of unweighted receiver choice to the TOA and TDOA algorithms yields 76% improvement of the median mean error for both. A 34% improvement in the median mean error is obtained by the addition of an unweighted receiver choice to the DD algorithm. The median standard deviations of the errors are improved by 91%, 91%, and 75%, respectively.

There are many possible lines of future work either to increase the level of realism of the model or extend what is known or can be done with the available data. These include more realistic target mobility models, implementation of tracking algorithms, and development of estimate quality metrics.

The work presented in this thesis is of both theoretical and practical significance. A new method of receiver choice for geolocation is proposed. Applying this proposed method to three different geolocation estimators in the context of a simulated IEEE 802.16 OFDMA network yields substantial improvement in performance. Finally, the proposed methods and the associated MATLAB package provide a starting point for a wide range of future research in this area.

ACKNOWLEDGMENTS

For friends and allies,

To mentors and editors;

My thanks forever.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Refinements to methods for passive geolocation of emitters, where these emitters are taken to be mobile devices in an Institute for Electrical and Electronics Engineering (IEEE) Standard 802.16 for Local and metropolitan area networks–Part 16: Air Interface for Broadband Wireless Access Systems (IEEE 802.16) compliant wireless communications network, are developed in this work [1]. This work follows that of [2] through both refinement and extension. Refinement is achieved through incorporation of clear air physical effects on the speed of propagation of radio frequency waves through the atmosphere. Two extensions are proposed. The first is by using the principles of the Doppler effect to compute estimated relative motion in addition to location. The second is to develop schema for exploiting choice of observing receiver to improve the performance of the basic position and velocity estimators. In this thesis the device to be located is known as a transmitter unless it must have some IEEE 802.16 OFDMA-specific feature, in which case it is known as a mobile station (MS). Likewise, the network equipment at which data about the transmitter is collected is known as receivers unless it must have some IEEE 802.16 OFDMA-specific feature, in which case it is known as a base station (BS).

A. BACKGROUND

The ability to locate a cellular handset or mobile station is one of growing importance in a number of areas. Central to all of them is that knowing the cellular handset's location is critical to the provision of some service. Two major areas in the civilian sector in which the ability to infer cellular handset's location information are emergency services and non-emergency location-based services. In either use case, it is clear that better position accuracy and precision contribute to the efficacy and relevance of the provided services.

The first case concerns provision of emergency services. Both the United States and the European Union have active regulatory efforts related to the ability to locate a cellular phone [3]. In the United States, this process is under the purview of the Federal Communications Commission (FCC) though the Enhanced 911 (E911) set of regulations. The corresponding set of European regulations is known as E112 [4]. Australia in recent years has implemented a National Emergency Warning System using text messages delivered to cell phones [5]. The November 1999 FCC E911 regulations specify maximum radial error requirements for handset location in order to facilitate effective provision of emergency services. E911 requires position estimate error of less than 50.0 m using handset-based or 100.0 m using network-based techniques for at least 67 percent of callers. Further, 95 per cent of callers are required to be able to be located to within 150 m using handset-based or 300 m using network-based techniques [3], [4]. While Australia's system for mass public alerts does not hinge on high accuracy position information, there is no reason to believe that the E911 or E112 regulations will not gradually specify progressively higher degrees of both accuracy and precision and, thereby, continue to drive the need for better geolocation systems.

Additionally, there are a growing number of non-emergency location based services. Some of the more well known are GM's OnStar and Mercedes-Benz's TeleAid systems [3]. Another major use has been the provision of yellow page services which provide the user with location specific information about nearby businesses [6]. Related to this use is interactive map consultation as performed on mobile phones [7]. Others include location-sensitive billing, for instance to provide billing rates for wireless access depending on whether the wireless terminal is used at home, in the office, or on the road [8]. Another is monitoring various at risk populations such as the mentally impaired [9], young children or parolees [10]. Location information could also be used for intelligent transportation systems and to enhance cellular network performance [11]. Currently, a number of commercial efforts are underway to exploit knowledge of a user's location for marketing purposes [10], [12].

B. RELATED WORK

A time difference-of-arrival (TDOA) method developed in [13] incorporates a set of parameters that account for the index of refraction in the line-of-sight from the transmitter to each observing receiver. The use of a single, common value for the index of refraction is proposed in this thesis.

A time-of-arrival (TOA) method for two-dimensional geolocation is presented in [14]. It uses the intersections of distinct circles to form linear constraints in the plane which may be used to estimate the position. An extension of this method to three dimensional geolocation is proposed in this thesis.

The method in [13] makes predetermined use of the information from five observing receivers. The algorithm in [14] suggests either using least-squares when extra constraints are available or computing a set of position estimates and applying some auxiliary algorithm to reduce them to a single-point estimate. The choice of a "good" subset from the total set of available constraint equations and then using the simplest version of the estimator to form the estimate is proposed in this thesis.

Both [13] and [14] require range estimates (which may be noisy) but do not explicitly consider range estimates with systematic bias. The former merely used an index of refraction-like parameter to account for additional delay due to channel characteristics [13]. Koorapaty, et al., consider biased range estimates but do not tie the degree of bias to a model of physical effects [15]. Explicitly considering biased range estimates, where the bias is tied to the mismatch between the estimated index of refraction of the medium and the true index of refraction of the medium, is proposed in this thesis.

C. OBJECTIVES AND APPROACH

The objective of this thesis is to improve the performance of geolocation schema. This is accomplished through estimating the speed of light, estimating target velocity, and exercising receiver choice. A flowchart of the overall scheme is shown in Figure 1. Exogenous inputs are denoted by parallelogram blocks. Rectangular blocks denote processes.

An estimate of the speed of light in the medium via inclusion of the Refractivity Estimation block and all the blocks that form its inputs is incorporated in this thesis. The proposed method for estimating target velocity is incorporated through the Target Velocity Estimation block. Finally, the proposed scheme for exploiting receiver choice is embodied in the Receiver Choice block. The Monte Carlo simulation undertaken to assess the effects of receiver choice on the performance of the various estimator is embodied in the User Application block.

D. ORGANIZATION

Background material, including existing theoretical work, is presented in Chapter II. This includes a discussion of the theory of circular and hyperbolic multilateration, the Doppler effect, the speed of light in an arbitrary medium, and those portions of the IEEE 802.16 specification necessary for the remainder of this thesis.

The biases that arise directly from assuming an incorrect value of the refractivity of the atmosphere are developed in Chapter III. There are three such biases. These are a propagation velocity bias, a range estimate bias, and a velocity estimate bias.

Methods for exploiting constraint choice in the contexts of TOA, TDOA, and Doppler velocity estimation (DVE) are presented in Chapter IV. After introducing some terminology, a general constraint choice algorithm is suggested based on simple linear algebra considerations. After this are discussions of how to apply this algorithm to the specific cases of TOA, TDOA and DVE. In the case of the TDOA problem, a modified estimator is derived to facilitate implementation of the technique.

The simulation study undertaken to prove the concepts developed in the preceding chapters is presented in Chapter V. The MATLAB package developed to facilitate simulation is documented at a high level. After this are sections in which the details of the various scenarios and results are presented.Finally, there is a discussion of the results.

The thesis is concluded in Chapter VI. A summary of the major work and suggestions for future work are presented.

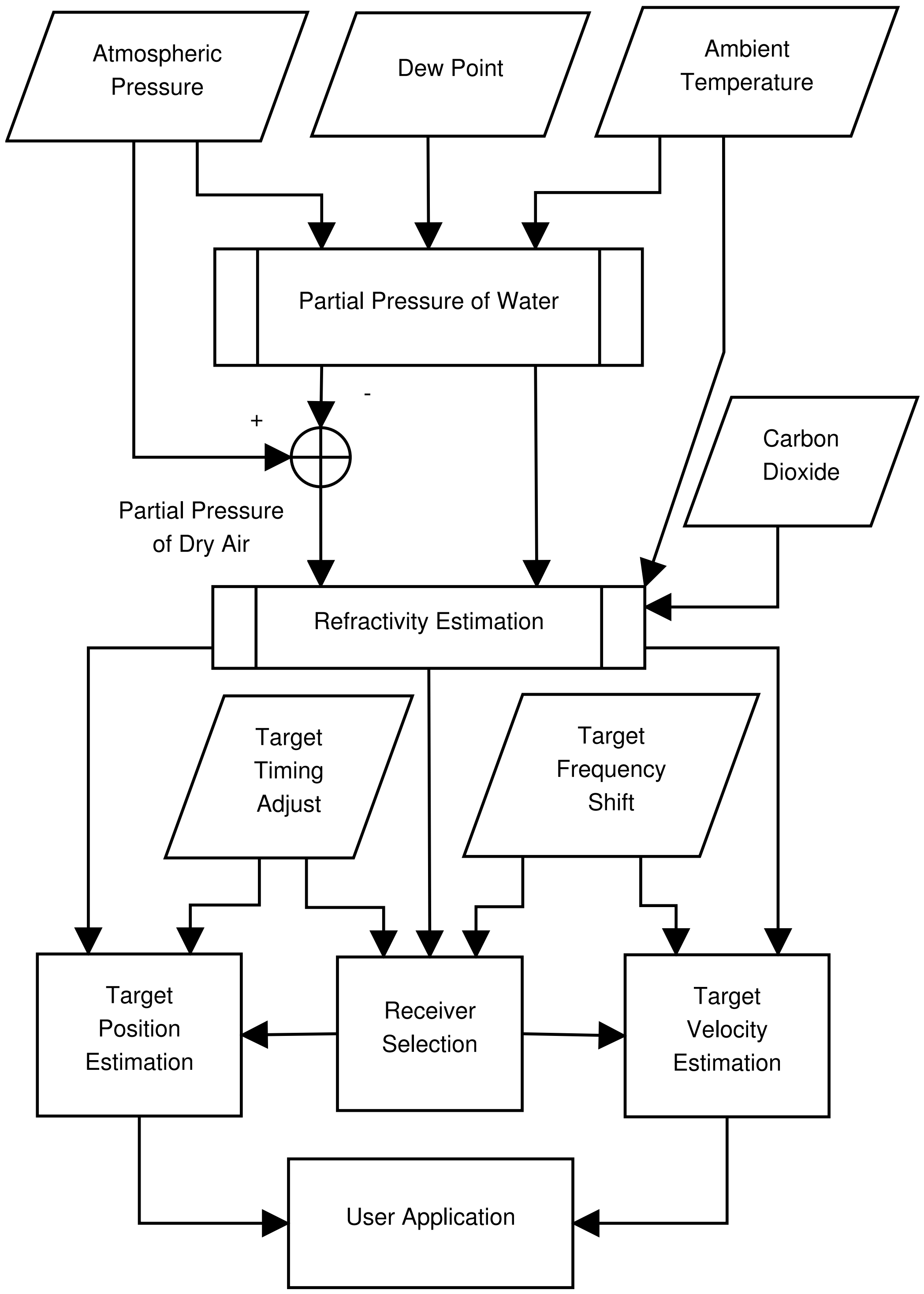

Figure 1. Flowchart of the overall geolocation estimation process including refractivity estimation and receiver choice.

There are three appendices to this thesis. A brief introduction to the Unified Modeling Language conventions used in this thesis to describe the MATLAB software developed as part of the work is presented in Appendix A. All of the MATLAB code which implements the various pieces of the software model is contained in Appendix B. Examples of the scripts used to conduct the simulations and generate the plots of the results are contained in Appendix C.

II. THEORY AND BACKGROUND

Presented in this chapter are the requisite theory and background information to support the remainder of this thesis. The related literature is presented in Section A. The theory behind geolocation of an emitter of interest using range information is presented in Section B. In Section C, the theory of how to use frequency shift information to estimate the velocity vector of an emitter is presented. Refractivity, refractive index, and the speed of light are discussed in Section D. Methods to estimate the refractivity of air are discussed in Section E. Finally, the relevant features of the IEEE 802.16 standard are presented in Section F.

A. GEOLOCATION OF EMITTERS

The literature related to passive geolocation of emitters is extensive. Survey papers aside, there are at least three major axes along which the literature may be classified. The first is whether the propagation paths are line-of-sight (LOS) or not (NLOS). The second relates to the information used to form estimates, be it propagation time, received frequency, angle, or in some cases other information. The third is the method of solution. Categories include least-squares, maximum-likelihood, constrained optimization, and geometric [16], [17]. Within this constellation of combinations, LOS techniques using propagation time are of the most immediate interest across the set of solutions of the resulting equations.

Given the breath and depth of the geolocation literature, several works are of great use as entry points. An excellent introduction to the relevant statistical theory of passive geolocation of emitters may be found in [18]. Papers that survey methods of geolocation applied to cell phones include [4], [6], [8], [19].

Propagation time based approaches dominate the literature. They may be further subdivided into two sub-classes. First are those that use the time to propagate from the emitter to each of a set of observers and is known as the time-of-arrival. The second are those that use the difference in time-of-arrival of a signal at pairs of different observers or the time difference-of-arrival. Each is discussed in the following.

Time-of-arrival approaches are also known as circular approaches because estimates are formed using the intersection of circles centered at each observer. One simple geometric approach uses pairs of overlapping circles to form lines of position, the intersection of which is taken to be the estimated emitter position [14]. This technique is applied in [16] to timing adjust (TA) information employed in synchronizing handset transmissions with the base station. In the case of unsynchronized transmitters and receivers, an approach to both localization and tracking is provided in [20].

Time difference-of-arrival approaches are also known as hyperbolic approaches because estimates are formed from the intersection of hyperbola with foci at the location of each of the two observers. The dual problem, that of navigation, is formulated and solved in [21]. A divide and conquer approach to solving the TDOA equations is presented in [22], but this approach is now somewhat dated. The case of fixed terrestrial transmitters and observers is studied in [23]. Satellites observing terrestrial emitters is studied in [24] and [25]. A generalized version incorporating both moving emitters and receivers is formulated in [26]. These papers contrast with [17] where the emitter is assumed to be in the far-field and employs linear approximations to the asymptotes of the hyperbolic curves to form position estimates or when in the near-field to seed other techniques. It is also possible to cast the problem as one of constrained optimization by considering an additional set of geometric constraints [27].

A few papers have explored the comparative merits of TOA versus TDOA or alternatively attempted to address bias problems caused by a variety of factors. The accuracy of TOA versus TDOA is studied in [28], which finds that circular techniques perform as well or better than hyperbolic techniques. While most papers consider the problem of noisy measurements to some degree, a systematic attempt to consider biased measurement information is undertaken in [15] using biases up to 200 m over ten km. A single paper has attempted to directly address the issue of propagation delay in the channel [13]. While using an idea similar to the index of refraction, a very general model is formulated in this paper but does not explicitly consider refractivity.

A few papers that fall outside of the rubric of LOS TOA or TDOA deserve mention. First are two papers that consider the use of frequency to form geolocation estimates [25, 26]. These papers conclude that while frequency information is a possible alternative, it adds significant computational complexity. Another frequency approach is that of differential Doppler [29]. A third family of geolocation approaches uses received signal strength, to which [30] serves as a useful starting point. The challenges of NLOS environments are discussed in [31], [32], [33].

B. GEOLOCATION USING RANGE ESTIMATES

As discussed in Chapter I, there is a continuing interest in using information about the location of a mobile device to provide a variety of services. Given a set of observations of the time required for a transmitted radio signal to propagate from the source to each of a set of receivers (RCVR), the task becomes to use this information to construct a position estimate in three spatial dimensions.

1. Spatial Model

The spatial model presented in this section is based on ranges from a set of observing receivers. Developed for scenarios in three dimensions, it follows a number of references: [2], [17], [14], [21], [23], [24], [25]. The model used in this chapter is shown in Figure 2.

Distance is related to propagation time via

$$r_i = \frac{c}{n} t_i \tag{1}$$

where r_i is the distance in meters from the transmitter to the i-th receiver, c is the speed of light in vacuum in meters per second, n is the refractive index, and t_i is the propagation

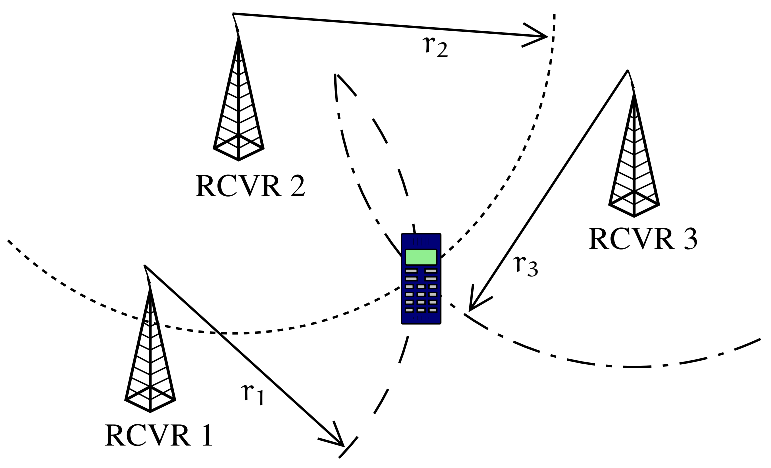

Figure 2. Spatial model of mobile subscriber position as the intersection of three range circles.

time in seconds to the i-th receiver. The refractive index is unit-free and accounts for the difference between the speed of light in air and vacuum.

It is necessary to assume the reference clocks are synchronized. The standard recommends synchronization to a common time reference signal; e.g., as provided by a Global Positioning System (GPS) receiver [1, 8.4.10.1.1].

2. Time of Arrival Algorithms

Time-of-arrival algorithms are also known as circular multilateration algorithms. This is because they act on the basis of intersecting circles with radii derived from the propagation time of a signal from the transmitter to a set of observing receivers. In the context of an IEEE 802.16 network, the transmitter may be taken to be a MS and each receiver to be a BS. The number of these towers is a function of the number of dimensions in which localization is desired.

An extension of the algorithm given in [14] may be derived in a straightforward manner. Begin by denoting the position of the transmitter of interest as p_0 and the positions of the various receivers as p_i where $i \in \mathbb{N}$. Assuming that the receivers are time-synchronized, we see that two equivalent formulations are possible. One further assumes that the time-of-arrival of a signal from the transmitter at each receiver is measured and the time of transmission is known. The other assumes that a measurement of the propagation time t_i from the transmitter to receiver i is known for each i. This work takes the second framework. Then we can write a set of circular constraints with the form

$$|p_i - p_0| = \nu_p t_i = r_i \tag{2}$$

where ν_p is the local propagation velocity in the medium and $|\cdot|$ indicates the absolute value of a scalar or the norm of a vector as appropriate. In three dimensions, the constraints described by (2) are spherical. Intersecting a pair of such spherical constraints gives a circular constraint in three dimensions. This circle is centered at point p_{ij} with radius r_{ij} where i and j are the indices of the receivers involved. Denote the distance between the center of the constraint circle and receivers i and j as $d_i = |p_{ij} - p_i|$ and $d_j = |p_{ij} - p_j|$, respectively. Invoking the Pythagorean theorem, we get the relationship

$$r_i^2 - d_i^2 = r_j^2 - d_j^2. \tag{3}$$

As p_{ij} lies on a line through p_i and p_j, then

$$p_{ij} = p_i \pm d_i \frac{p_j - p_i}{|p_j - p_i|}. \tag{4}$$

The $\pm$ in (4) is because $d_i > 0$.

The simplest way to solve for the intersections of the two spheres is by considering three cases. Without loss of generality, suppose that the line connecting p_i and p_j is oriented such that p_i is to the left of p_j. Then the three cases are that p_{ij} is to the left of p_i, between p_i and p_j, and to the right of p_j. It will be shown that these three can be reduced to the first two cases. These cases are depicted in Figures 3, 4, and 5, respectively.

In the first case, rewrite (3) as

$$r_i^2 - d_i^2 = r_j^2 - \left(d_i + |p_j - p_i|\right)^2. \tag{5}$$

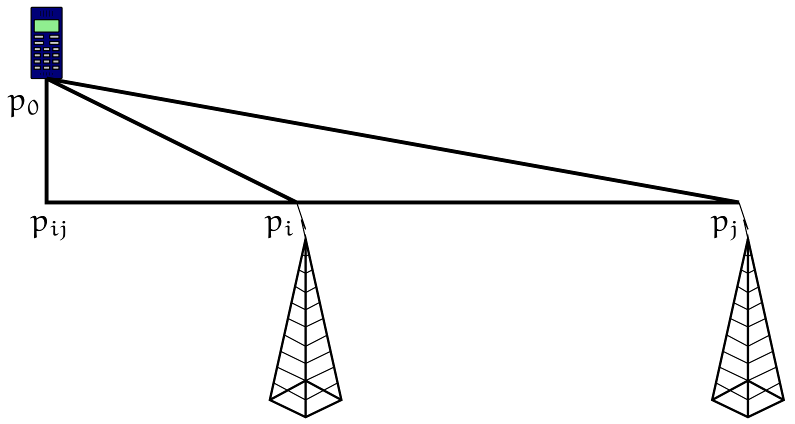

Figure 3. The case of p_{ij} to the left of p_i.

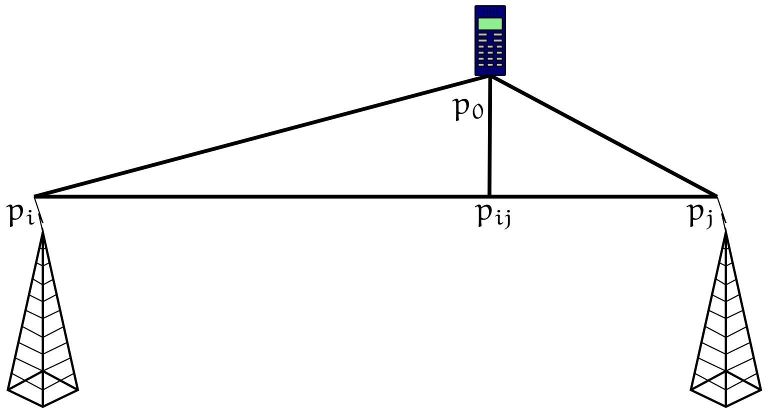

Figure 4. The case of p_{ij} in between p_i and p_j .

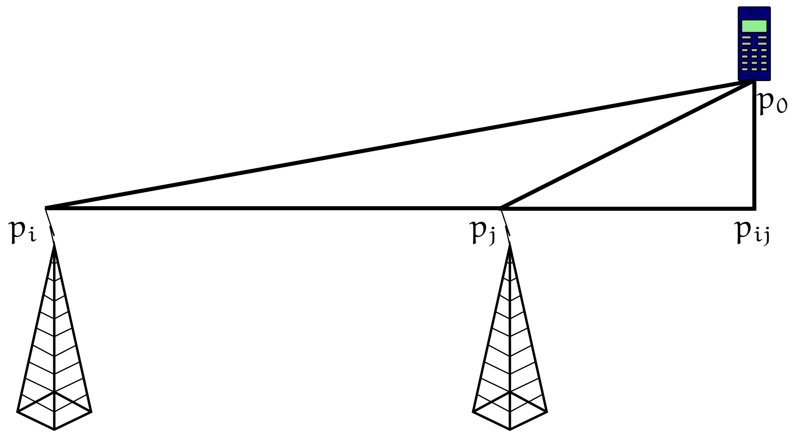

Figure 5. The case of p_{ij} to the right of p_i and p_j .

Expand the right hand side to obtain

$$r_i^2 - d_i^2 = r_j^2 - d_i^2 - 2d_i|p_j - p_i| - |p_j - p_i|^2. \tag{6}$$

Finally, solve for d_i to produce the relationship

$$d_i = \frac{r_i^2 - r_j^2 + |p_j - p_i|^2}{-2|p_j - p_i|}. \tag{7}$$

In this case, the position of p_{ij} is given by

$$p_{ij} = p_i - d_i \frac{p_j - p_i}{|p_j - p_i|}. \tag{8}$$

In the second case, rewrite (3) as

$$r_i^2 - d_i^2 = r_j^2 - \left(|p_j - p_i| - d_i\right)^2. \tag{9}$$

Expand the right hand side to obtain

$$r_i^2 - d_i^2 = r_j^2 - d_i^2 + 2d_i|p_j - p_i| - |p_j - p_i|^2. \tag{10}$$

Finally, solve for d_i to produce the relationship

$$d_i = \frac{r_i^2 - r_j^2 + |p_j - p_i|^2}{2|p_j - p_i|}. \tag{11}$$

In this case, the position of p_{ij} is given by

$$p_{ij} = p_i + d_i \frac{p_j - p_i}{|p_j - p_i|}. \tag{12}$$

In the third case, rewrite (3) as

$$r_i^2 - d_i^2 = r_j^2 - \left(d_i - |p_j - p_i|\right)^2. \tag{13}$$

This is facially equivalent to the second case.

These three circumstances are summarized Table 1. These three cases may be unified by defining a new variable d such that

$$d = \begin{cases} -d_i & \text{Case 1} \\ d_i & \text{Case 2, 3.} \end{cases} \tag{14}$$

This allows (4) to be simplified to

$$p_{ij} = p_i + d\frac{p_j - p_i}{|p_j - p_i|}. \tag{15}$$

Table 1. Summary of the TOA solutions.

Case	d_i	p_{ij}
1	$\frac{r_i^2 - r_j^2 + \lvert p_j - p_i\rvert^2}{-2\lvert p_j - p_i\rvert}$	$p_i - d_i\frac{p_j - p_i}{\lvert p_j - p_i\rvert}$
2	$\frac{r_i^2 - r_j^2 + \lvert p_j - p_i\rvert^2}{2\lvert p_j - p_i\rvert}$	$p_i + d_i\frac{p_j - p_i}{\lvert p_j - p_i\rvert}$
3	$\frac{r_i^2 - r_j^2 + \lvert p_j - p_i\rvert^2}{2\lvert p_j - p_i\rvert}$	$p_i + d_i\frac{p_j - p_i}{\lvert p_j - p_i\rvert}$

It is now possible in three dimensions to use four towers to form three circular constraints and, using these constraints, form three planar constraints, which may then be solved. This derivation will rely on the fact that an arbitrary plane in $\mathbb{R}^3$ may be described by

$$\vec{n}^\top (x - x_0) = 0 \tag{16}$$

where $\vec{n}$ is any vector normal to the plane, x is the position of any point in the plane expressed as a column vector, and x_0 is the position of an arbitrary reference point in the plane expressed as a column vector. Let three non-co-planer circular constraints be formed from four towers using the method described above. Further, let their centers be located without loss of generality at ψ_i such that $i = 1, 2, 3$ and their radii be ρ_i such that

$i = 1, 2, 3$. Then these three circles may be written as

$$|p_0 - \psi_1| = \rho_1 \tag{17}$$

$$|p_0 - \psi_2| = \rho_2 \tag{18}$$

$$|p_0 - \psi_3| = \rho_3. \tag{19}$$

As the circles described by (17) through (19) are in $\mathbb{R}^3$, their respective orientation vectors may be taken to the the unit vector normal to the plane in which the circle lies. Recalling (16), we choose

$$\vec{n}_{ij} = \frac{p_j - p_i}{|p_j - p_i|}. \tag{20}$$

Next, recognizing the intersection of the three constraint circles lies in the three planes, the problem of solving for the intersection of three circles is now reducible to solving the intersection of three planes with the form

$$\vec{n}_{ij}^{\top}(x - p_{ij}) = 0. \tag{21}$$

Congruent to the convention of using ψ_i to represent the center of a circle, let $\vec{n}_i$ be the corresponding normal vector. Ergo, the final set of linear equations to solve has the form

$$\vec{n}_1^{\top} p_0 = \vec{n}_1^{\top} \psi_1 \tag{22}$$

$$\vec{n}_2^{\top} p_0 = \vec{n}_2^{\top} \psi_2 \tag{23}$$

$$\vec{n}_3^{\top} p_0 = \vec{n}_3^{\top} \psi_3. \tag{24}$$

There are two special cases which lead to degeneracy. The first is the case of three co-linear towers. In this case, any two pairs will form the same circular constraint. This is clear from considering the geometry of the single circle case in that the transmitter is always a fixed radius from the line connecting two receivers regardless of how far those receivers may be from the target or each other; therefore, the circular constraint is the same both in radius, center, and orientation. The second case is of four co-planar receivers. In this case, taking any three planar constraints, it may be seen that the normal vector to the plane of the observing towers is parallel to every line in any plane. This is equivalent to the three planes being linearly dependent.

3. Time Difference of Arrival Algorithms

Time difference-of-arrival methods proceed from using the observed reception time at pairs of time-synchronized receivers to form a set of hyperbolic constraints in $\mathbb{R}^3$, which, when intersected, yield a position estimate. This approach has the advantage of not requiring knowledge of the time of transmission of the received signal [13]. These hyperbola are defined by the difference in the distance from a transmitter to each of two distinct observing receivers. Like TOA algorithms, the number of receivers required is a function of the number of dimensions in which localization is desired. An example of data from three receivers forming two hyperbola is shown in Figure 6. The hyperbola curves toward the receiver to which the transmitter is closer.

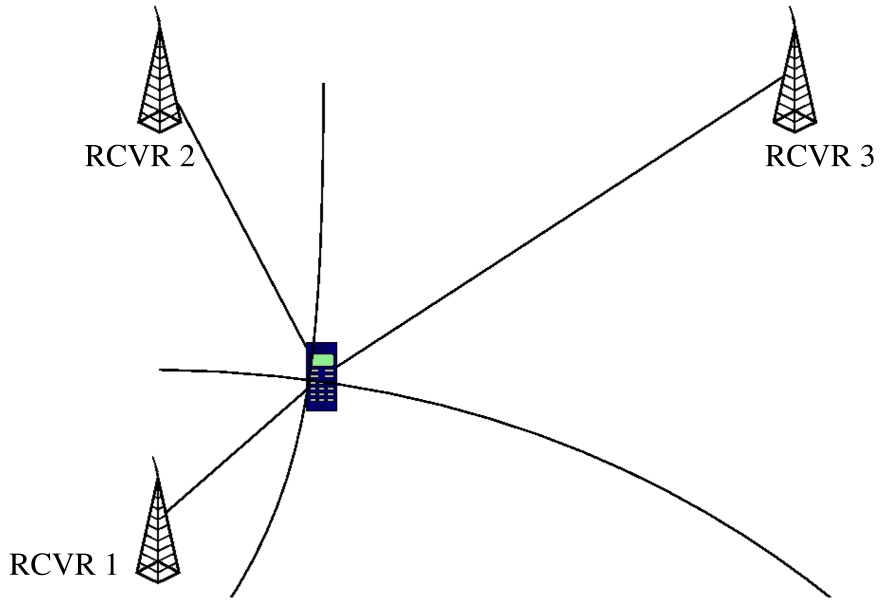

Figure 6. An example of range differences from three receivers forming two hyperbola, the intersection of which is the estimated position of the transmitter.

The base scheme used in this work is due to Bakhoum [13]. It is attractive because it reduces the problem of hyperbolic estimation to solving a matrix equation with the form

$$\mathbf{A}p_0 = \mathbf{b} \tag{25}$$

where the i-th row of $\mathbf{A}$ is given by

$$\mathbf{A}_i = \frac{2}{t_2 - t_1}\left(\frac{\vec{p}_2^\top}{\alpha_2^2} - \frac{\vec{p}_1^\top}{\alpha_1^2}\right) - \frac{2}{t_{i+2} - t_1}\left(\frac{\vec{p}_{i+2}^\top}{\alpha_{i+2}^2} - \frac{\vec{p}_1^\top}{\alpha_1^2}\right) \tag{26}$$

and the corresponding element of $\mathbf{b}$ by

$$\mathbf{b}_i = \frac{1}{t_2 - t_1}\left(\frac{|\vec{p}_2|^2}{\alpha_2^2} - \frac{|\vec{p}_1|^2}{\alpha_1^2}\right) - \frac{1}{t_{i+2} - t_1}\left(\frac{|\vec{p}_{i+2}|^2}{\alpha_{i+2}^2} - \frac{|\vec{p}_1|^2}{\alpha_1^2}\right) + c^2(t_{i+2} - t_2). \tag{27}$$

The time-of-arrival of the signal at BS i is t_i, the position of BS i is $\vec{p}_i$, and the reciprocal of the refractive index along the line-of-sight between the emitter and the BS is α_i.

One important consideration for this scheme is that it will fail whenever there is no difference in the time-of-arrival of a signal at two different BS. This results in division by zero and a consequent failure to produce a meaningful position estimate. If the time measurements are sufficiently precise, then it becomes almost impossible for such a condition to occur. As will be discussed in more detail in Section F, time measurements in the context of an IEEE 802.16 network do not have the level of precision required to preclude the possibility of division by zero. One method for addressing this problem is presented in Chapter IV.

C. VELOCITY ESTIMATION FROM FREQUENCY INFORMATION

In addition to estimating the location information using techniques presented in Section B, it is also possible to estimate the motion of the MS using the Doppler Equation and the observed frequency shift at a set of BS. The Doppler equation and the spatial model of mobile device motion are introduced in Section 1. A method for using the Doppler equation to solve for the velocity of the MS is presented in Section 2.

1. Spatial Model and the Doppler Equation

The Doppler equation relates relative motion of a transmitter with respect to a receiver to the ratio of the received and transmitted frequencies. When the relative motion between the transmitter and the receiver is small relative to the propagation velocity, as is the case here, the received frequency f is related to the original frequency f_0 as

$$f = f_o(1 - \beta) \tag{28}$$

where $\beta = v/c$ and v is the radial speed assuming the transmitter and receiver are moving apart from each other [34, Eq. 37-33, p. 1040]. It is convenient for the purposes of this work to re-express the Doppler relationship between receiver i and the transmitter of interest as

$$f_i = f_t \left(1 - \frac{v_i}{v_p}\right) \tag{29}$$

where f_i is the frequency received at receiver i, f_0 is replaced with the transmitted center frequency f_t, v_i is the signed relative motion in the line-of-sight between the transmitter and the receiver in meters per second, and v_p is the velocity of signal propagation in the medium in meters per second. Note that the sign convention is that positive v_i indicates the transmitter is opening the receiver in range resulting in the expected decrease in observed frequency.

In the context of cellular networks, the only unknowns in (29) are v_i and v_p. The former is discussed below. The latter may be estimated from physical considerations. In this section, it is assumed that a reasonable estimate exists, and we will take up the question of how to create such an estimate in Section D.

In order to apply the Doppler equation to the problem at hand, it is necessary to have a spatial model of the motion of the mobile device. This is done with the use of

Hamiltonian notation to develop a single equation and then is shifted to matrix notation in the final solution.

We first define the new notations used in the model. Let $\vec{O}$ be the arbitrary origin of the system with components measured in meters. Let $\vec{v}(t)$ be the transmitter's velocity as a function of time with components measured in meters per second. This is denoted by $\vec{v}$ where confusion will not result. Finally, let

$$\vec{n} = \frac{p_0 - p_i}{|p_0 - p_i|} \tag{30}$$

be a unit-less unit vector in the direction of p_0 relative to receiver p_i. The vector $\vec{n}$ is pointed away from p_i to conform with the convention above that positive relative motion between the transmitter and receiver indicates opening in range. It follows immediately that, at any time t, the relative motion in the line of sight $\vec{v}_i$ between the transmitter and receiver is given by the inner product of $\vec{n}$ and $\vec{v}$

$$\vec{v}_i = \frac{(p_0 - p_i) \cdot \vec{v}}{|p_0 - p_i|}. \tag{31}$$

This situation is shown in Figure 7. The vector $\vec{n}$ has been drawn in bold over $\vec{v_i}$, and both vectors begin at the same location.

2. Doppler Velocity Estimation with a Known Transmitted Frequency

With the information developed above, it is now possible to derive an estimator for the motion of the transmitter. This estimator is derived below in two steps. First, (29) and (31) for two receivers are used to form a linear relationship between observed frequencies f_i and f_j ($i \neq j$) and $\vec{v}$ as given in (33). Second, an estimate of $\vec{v}$ is created by rewriting the aforementioned equation for multiple independent receiver pairs in matrix form and solving for $\vec{v}$.

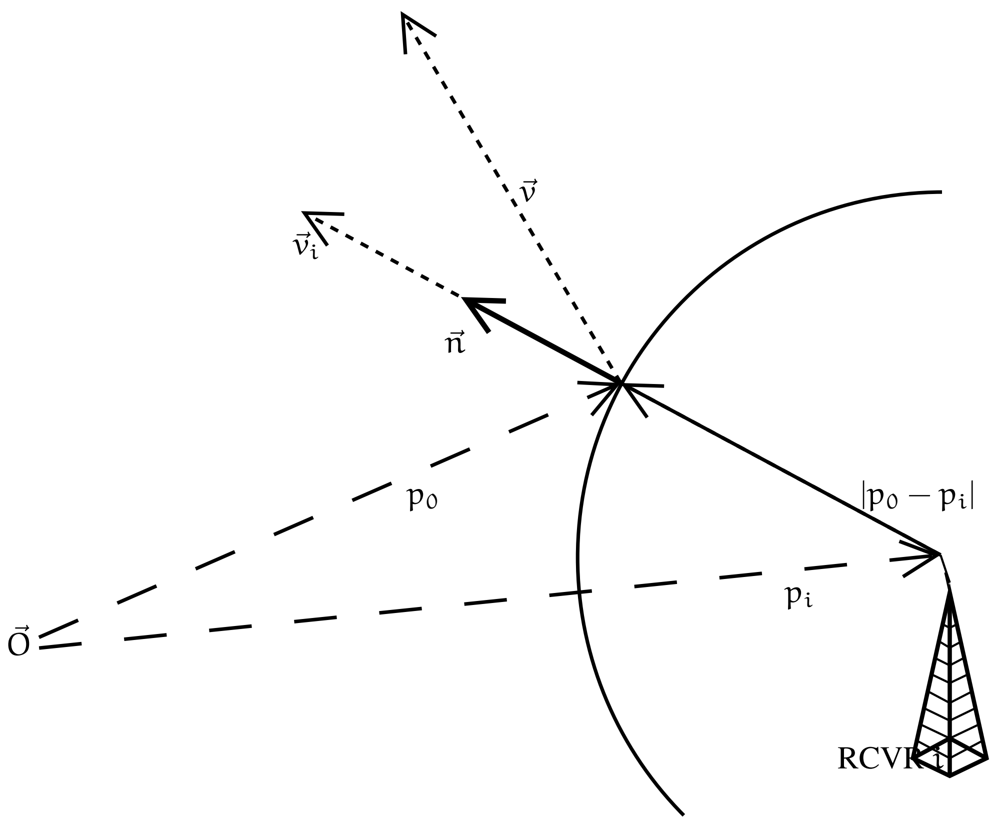

Figure 7. Model of transmitter motion with a nominal velocity vector indicating the transmitter is moving away from the receiver.

Substituting (31) into (29), we get

$$\begin{aligned} f_i &= f_t \left(1 - \frac{\frac{(p_0 - p_i)\cdot\vec{v}}{|p_0 - p_i|}}{v_p}\right) \\ &= f_t \left(1 - \frac{(p_0 - p_i)\cdot\vec{v}}{v_p|p_0 - p_i|}\right) \\ &= f_t \left(1 - \frac{(p_0 - p_i)\cdot\vec{v}}{t_i v_p^2}\right). \end{aligned} \tag{32}$$

The last step follows from recalling $|p_0 - p_i| = t_i v_p$ where t_i may be taken to be the timing adjust value associated with receiver i. In general, t_i may be any estimate of the propagation time of a transmission from the transmitter to the i-th receiver.

Next, assuming available data for four receiving base stations, we form the frequency difference relative to p_i by taking $f_{j,i} = f_j - f_i, \; i = 1, 2, 3$ for the three-dimensional case. This yields

$$\begin{aligned} f_{j,i} &= f_j - f_i \\ &= f_t\left(1 - \frac{(p_0 - p_j)\cdot\vec{v}}{t_j v_p^2}\right) - f_t\left(1 - \frac{(p_0 - p_i)\cdot\vec{v}}{t_i v_p^2}\right) \\ &= f_t\left(1 - \frac{(p_0 - p_j)\cdot\vec{v}}{t_j v_p^2} - 1 + \frac{(p_0 - p_i)\cdot\vec{v}}{t_i v_p^2}\right) \\ &= f_t\left(\frac{p_j\cdot\vec{v}}{t_j v_p^2} - \frac{p_i\cdot\vec{v}}{t_i v_p^2}\right) \\ &= \frac{f_t}{v_p^2}\left(\frac{p_j}{t_j} - \frac{p_i}{t_i}\right)\cdot\vec{v} \end{aligned} \tag{33}$$

Note that the values for f_j and f_i must correspond in time.

Next, without loss of generality, let $i = 1$ and $j = 2, 3, 4$ in (33) to form

$$f_{2,1} = \frac{f_t}{v_p^2}\left(\frac{p_2}{t_2} - \frac{p_1}{t_1}\right)\cdot\vec{v} \tag{34a}$$

$$f_{3,1} = \frac{f_t}{v_p^2}\left(\frac{p_3}{t_3} - \frac{p_1}{t_1}\right)\cdot\vec{v} \tag{34b}$$

$$f_{4,1} = \frac{f_t}{v_p^2}\left(\frac{p_4}{t_4} - \frac{p_1}{t_1}\right)\cdot\vec{v}. \tag{34c}$$

This is the minimum case to generate a unique estimate of $\vec{v}$ assuming motion in three dimensions. In the event the system is under-determined, additional knowledge must be brought to bear in forming a reasonable estimate. In the event that there are more than four receivers available, then the system is over-determined, and techniques such as weighted least squares may be applied.

Rewriting the left hand side of (34a) through (34c) as a column vector $\mathbf{f}_{ji}$, $\vec{v}$ as the column vector $\mathbf{v}$ and $p_j/t_j - p_1/t_1$ as a three by three matrix $\mathbf{R}_{j,i}$, we get the simplified relationship

$$\mathbf{f}_{j,i} = \frac{f_t}{v_p^2}\mathbf{R}_{j,i}\mathbf{v}. \tag{35}$$

Assuming that $\mathbf{R}_{j,i}$ is invertable, we get the estimate of **v** as

$$\mathbf{v} = \frac{v_p^2 \mathbf{R}_{j,i}^{-1}\mathbf{f}_{j,i}}{f_t}. \tag{36}$$

This assumption is reasonable from the construction of $\mathbf{R}_{j,i}$ so long no as two receivers are co-linear with respect to the reference station.

D. REFRACTIVE INDEX AND REFRACTIVITY

The refractive index of air refers to the degree to which light is slowed by traveling through the atmosphere relative to traveling through vacuum [35]. This relationship is given mathematically as

$$c_n = \frac{c}{n} \tag{37}$$

where n is the refractive index, and c_n is the speed of light in the medium [34, Eq. 35-3, p. 960].

This thesis follows the geodesy literature by discussing the refractivity of light in a given medium. Refractivity N is related to refractive index as

$$N = 10^6(n-1) \tag{38}$$

expressed in parts per million (ppm) [36]. Ergo, while the refractive index expresses the true relationship between the the speed of light in vacuum and the speed of light in a given medium, the refractivity is a shorthand convention which is more convenient given the range of values in use (less than 1000 ppm).

E. ESTIMATING THE REFRACTIVITY OF AIR

The refractivity of the atmosphere is driven by local atmospheric conditions, most notably partial pressure of dry air, partial pressure of water, temperature, and carbon dioxide content [36]. The partial pressure of dry air is that part of total atmospheric pressure due to air alone (no water vapor). Partial pressure of water is that part due to the water vapor in the air and depends on the dew or frost point and overall atmospheric pressure [37]. The equation used to estimate refractivity is discussed first followed by a discussion of how to obtain the required partial pressure values from common meteorological data. The overall estimation process is shown in Figure 8.

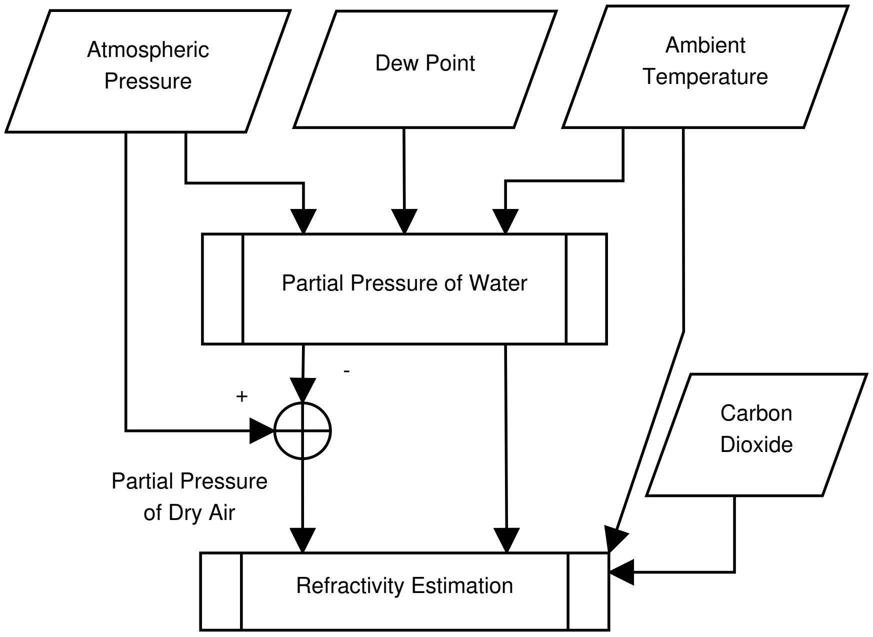

Figure 8. Flowchart of the process of estimating local refractivity from atmospheric conditions.

1. Empirical Formulae for Refractivity Estimation

The literature concerning the refractive index of air (also known as the refractivity of air) dates to at least 1933 [38]. Pencil and paper formulae provide good approximate values of atmospheric refractivity so long as the frequency of interest avoids certain critical values near 22, 60, and 120 GHz [36], [35]. It is assumed herein that the network operates below 11 GHz, thereby, entirely avoiding these problems.

Modern literature on modeling the refractivity of the Earth's atmosphere dates to the 1950s [36], [38]. During this time, the Effective Earth Radius model, also known as the "4/3 Earth Model," was supplanted by a model of long term refractivity, which incorporated ambient temperature, total atmospheric pressure, and water vapor pressure [36], [38], [39]. In the 1950s and into the 1960s, much of the data used for estimation of physical constants was taken from radiosonde observations [38], [39]. Between 1960 and 1963, the International Union of Geodesy and Geophysics settled on a standard model [36]. Various attempts to improve on these formulae and their use have been made in the intervening years, and the set of parameters has been expanded to explicitly include atmospheric carbon dioxide concentration [36], [40], [41]. In the context of GPS, using per observation refractivity adjustments has been shown to be significantly more effective and accurate than using daily average corrections. In the context of surveying using very long baseline interferometry, the situation is more complicated, but corrections should still be applied for accurate measurement [41].

Later work has refined the coefficients used in these models to reflect more recent data collected in lab settings using refractometers [36], [40], [42]. Comparison of the modern versions of these models indicates general agreement to within two ppm for most temperature and humidity conditions [36]. With the aforementioned considerations in mind, the "best average coefficients" formula for 375 ppm atmospheric carbon dioxide given by [36]

$$N = 77.6890\frac{P_d}{T + 273.15} + 71.2952\frac{P_w}{T + 273.15} + 375463\frac{P_w}{(T + 273.15)^2} \tag{39}$$

is adopted for use in the model such that P_d is the partial pressure of dry air in millibar with the specified amount of carbon dioxide, P_w is the partial pressure of water vapor in millibar, and T is the temperature in Celsius. A plot of refractivity as a function of temperature and dew-point spread is shown in Figure 9. One thousand millibar total atmospheric pressure is one standard atmosphere of pressure. While (39) is cast in terms of temperature and two partial pressures, it is possible to compute the partial pressures required from temperature and dew-point using equations discussed in Section E.2. Under most conditions (those in the lower right quadrant) refractivity increases with increasing temperature and decreased dew-point spreads. The warmest and wettest parts of the atmosphere have the highest refractivity values, while cold, dry areas have lower refractivity values. The discontinuity at zero Celsius is due to a discontinuity in the underlying formula for the partial pressure of water.

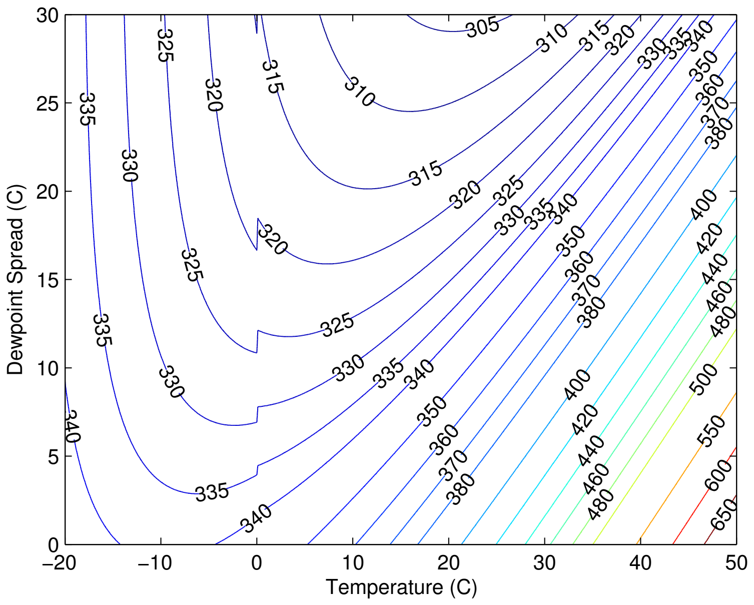

Figure 9. Refractivity of the atmosphere as a function of temperature and dew-point spread at 1000 millibar total atmospheric pressure.

A region may also undergo significant changes in refractivity due to daily and seasonal changes. One example is documented in [43]. Data taken for Akure, Southwestern Nigeria show daily average swings of 30 ppm during the winter months and an average of a 50 ppm difference in refractivity between summer and winter.

2. Partial Pressure of Dry Air and Water Vapor

In practice, it is reasonable to obtain local ambient temperature, total atmospheric pressure P, and dew- or frost-point (T_d and T_f, respectively) from meteorologists. All temperatures are measured in Celsius, and all pressures are measured in millibars. Using the equations and tables given in [37], we can to estimate the partial pressure of water vapor as

$$P_w = \begin{cases} e(T_d)f(T_d, P) & T \in (0, 100] \\ e(T_f)f(T_f, P) & T \in [-50, 0] \end{cases} \tag{40}$$

where $e(T)$ is the partial pressure of water vapor and $f(T, P)$ is an enhancement factor to account for behavioral differences between moist air and pure water vapor, given temperature and pressure. Several families of $e(T)$ curves that have been optimized for various temperature ranges and have associated optimal $f(T, P)$ curves are presented in [37]. The curves adopted for use in this thesis were chosen from Table 2 of [37] for a combination of their accuracy over the specified range of values and their relative simplicity. The partial pressure of water vapor curve adopted from [37] is given by

$$e(T) = \begin{cases} 6.1121 \exp\left\{\frac{(18.564-T/254.4)T}{T+255.57}\right\} & T \in (0, 100] \\ 6.1115 \exp\left\{\frac{22.452T}{T+272.55}\right\} & T \in [-50, 0]. \end{cases} \tag{41}$$

The associated enhancement factor adopted from [37] is

$$f(T, P) = \begin{cases} 1 + 7.2 \times 10^{-4} + P[3.2 \times 10^{-6} + 5.9 \times 10^{-10}T^2] & T \in (0, 100] \\ 1 + 3 \times 10^{-4} + 4.18 \times 10^{-6}P & T \in [-50, 0]. \end{cases} \tag{42}$$

Finally, the partial pressure of dry air $P_d = P - P_w$ [36]. The partial pressure of water in the atmosphere as a function of temperature and dew-point spread is shown in Figure 10; total pressure is assumed to be 1000 millibar (approximately one standard atmosphere [35, Eq. 6.9]). The partial pressure of water in the atmosphere increases with temperature and dew-point (reduction in dew-point spread). The discontinuity at zero Celsius is due to the coefficients for the two regions having been fit separately without the requirement that the curves join at that point.

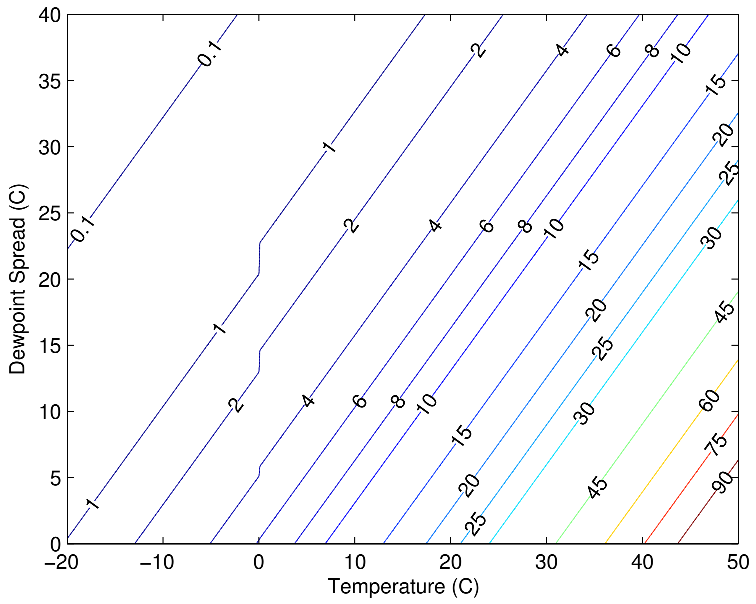

Figure 10. Partial pressure of water at 1000 millibars total pressure as a function of temperature and dew-point spread.

F. IEEE 802.16

1. OFDMA Waveform

The orthogonal frequency-division multiple access (OFDMA) physical layer (PHY) specification is an extension of the orthogonal frequency-division multiplex (OFDM) PHY. This extension supports scalability, multiple access, and advanced antenna array processing through division of each symbol into logical subchannels [1, 8.4.2.2]. Subchannels are comprised of groups of OFDM subcarriers. Subcarriers need not be adjacent in frequency. Each logical block is then independently assigned by the BS. During downlink, each MS is assigned a subchannel containing data addressed to it. Likewise, in uplink each MS is assigned a subchannel on which to transmit information to the BS.

An example of subcarriers assigned to various subchannels is shown in Figure 11. In this example, subcarriers assigned to each subchannel are nonadjacent. While this example shows only three subchannels, up to sixty are possible [1, Table 315].

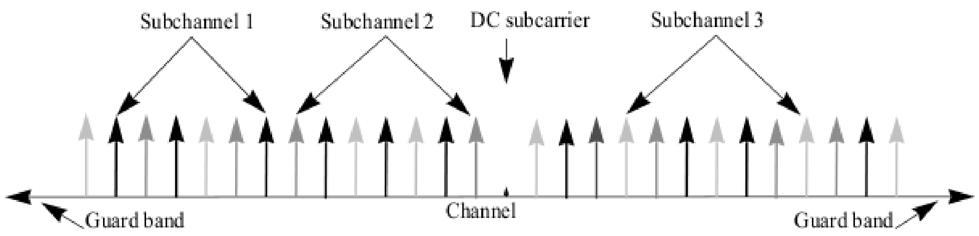

Figure 11. An OFDMA symbol in the frequency domain (three channel schematic example). From [1, Fig. 218].

An example of how an OFDMA frame might be subdivided in time-division duplexing mode is shown in Figure 12. The frame begins with a preamble. There is a frame

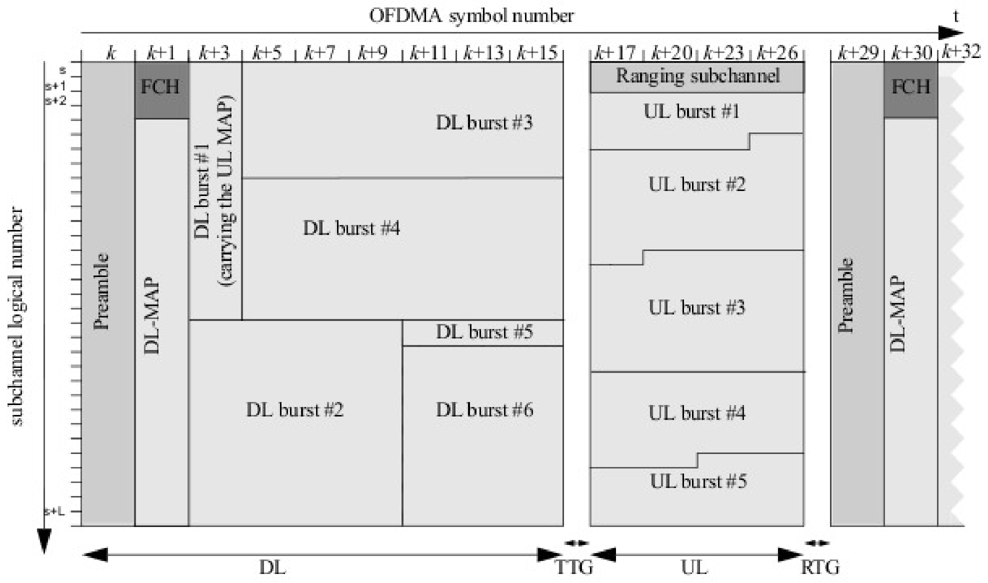

Figure 12. Example of an OFDMA frame (with only mandatory zone) in TDD mode. From [1, Fig. 222].

control header, a downlink map (DL-MAP), an uplink map (UL-MAP), and bursts allocated to each MS.

2. Network Specification

IEEE 802.16 OFMDA networks can be characterized into equivalence classes on any subset of four primitive parameters. The four primitive parameters are listed in Table 2 [1, 8.4.2.3]. These four primitive parameters are then used to derive additional parameters pursuant to the definitions given in chapter 8.4.2.4 of [1], as shown in Table 3.

Table 2. Primitive IEEE 802.16 OFDMA network parameters.

Parameter	Definition
Nominal channel bandwidth	W
Number of used subchannels	N_{used}
Sampling factor	$n = \begin{cases} 8/7 & W \mod 1.75\,\text{MHz} = 0 \\ 28/25 & W \mod X\,\text{MHz} = 0, \\ & X \in \{1.25, 1.5, 2, 2.75\} \\ 8/7 & \text{otherwise} \end{cases}$
Ratio of cyclic prefix time to useful time	$G \in \{1/32, 1/16, 1/8, 1/4\}$

Table 3. Derived IEEE 802.16 OFDMA network parameters.

Parameter	Definition
N_{fft}	Smallest power of two greater than N_{used}
Sampling Frequency	$F_s = 8000\lfloor n \cdot W/8000 \rfloor$
Subcarrier spacing	$\Delta f = F_s/N_{fft}$
Bit Time	$T_b = 1/\Delta f$
Cyclic Prefix Time	$T_g = GT_b$
OFDMA Symbol Time	$T_s = T_b + T_g$
Sampling Time	T_b/N_{fft}

3. Synchronization and Ranging

Ranging is defined as a collection of steps by which the quality of the radio frequency link between the subscriber station and BS is maintained [1, 6.3.10]. The ranging process encompasses a variety of parameters, two dedicated message structures, and three procedures. The process of ranging when using the OFDMA physical layer specification is defined in chapter 6.3.10.3 of [1].

Under the OFDMA physical layer specification, ranging for time and power occurs on a periodic basis, during (re)registration, and when synchronization is lost [1, 8.4.10.2]. Frequency adjustments are also transmitted by the BS to the MS as part of the ranging messages as necessary [1, 8.4.15.1]. Periodic ranging opportunities are controlled by the MS and must occur at least every 35 seconds [1, 6.3.10.3.2, Table 554].

The complete set of parameters and values which must or may be present in a Range Response (RNG_RSP) message are given in 6.3.2.3.6 in [1]. Of these, two are of interest. These are the Timing Adjust Information and the Frequency Adjust Information.

4. Extracting Time and Frequency Information

Extracting time and frequency information from the RNG_RSP message requires knowledge of both the field format specification and the associated units. This information is given in Table 585 of [1]. The timing adjust field is a signed 32-bit integer number of timing adjust units. A timing adjust unit is computed in seconds using the relationship given in 10.3.4.3 of [1]

$$\tau = \frac{1}{F_s} = \left(8000 \left\lfloor \frac{nW}{8000} \right\rfloor \right)^{-1} \tag{43}$$

where the equation for the sampling frequency is given in Table 3. The frequency adjust field is a signed 32-bit integer with units of Hz [1, Table 585].

Conversion of timing adjust units to a range estimate is through the relationship

$$r = \nu_p \tau T_a \tag{44}$$

where T_a is an integer number of timing adjust units.

Relevant background material was presented in this section. A summary of the geolocation literature was presented. Three geolocation methods, TOA, TDOA, and DVE, were introduced. For the TOA and DVE methods, the presentation included a derivation of the estimator. Refractivity of the atmosphere was shown to be estimable from commonly available atmospheric quantities using formulas found in the literature. It was shown that the necessary inputs to the various estimators may be extracted from IEEE 802.16 RNG_RSP packets.

THIS PAGE INTENTIONALLY LEFT BLANK

III. BIASES ARISING FROM REFRACTIVITY MISMATCH

Using an incorrect value for local atmospheric refractivity introduces three directly quantifiable biases. These are propagation velocity bias, range bias, and target velocity bias. Closed form equations for each are given below. Propagation velocity bias is presented in Section A. Range estimate bias is presented in Section B. Target velocity bias is presented in Section C.

A. PROPAGATION VELOCITY BIAS

An incorrect choice of assumed local refractivity biases the estimated propagation velocity of the medium. Letting n_0 be the true value of the refractive index of the medium and $\hat{n}$ be the estimate, we can rewrite (37) as

$$c_{\hat{n}} = \frac{c}{\hat{n}} + \frac{c}{n_0} - \frac{c}{n_0} = \frac{c}{n_0} + \frac{n_0 - \hat{n}}{n_0 \hat{n}} c. \tag{45}$$

In this formulation, the bias is given by the right-most term

$$b_c(n_0, \hat{n}) = \frac{n_0 - \hat{n}}{n_0 \hat{n}} c. \tag{46}$$

This bias term is the amount by which the the estimated propagation velocity is too high (positive sign) or too low (negative sign).

B. RANGE ESTIMATE BIAS

A refractive bias arises in range estimates when the estimated refractivity is different from the true refractivity. Given propagation time to the ith base station, the distance estimate to the same base station is given by

$$r_i = \frac{c}{n_0} t_i + \frac{n_0 - \hat{n}}{\hat{n} n_0} c t_i. \tag{47}$$

By inspection, it is clear that when $\hat{n} = n_0$, r_i is unbiased; otherwise, bias due to refractivity mismatch is

$$b_d(n_0, \hat{n}) = \frac{n_0 - \hat{n}}{\hat{n} n_0} c t_i. \tag{48}$$

Adjusting for bias is now as simple as using an estimate of refractivity obtained via methods presented in Section II.E. Range bias as a function of propagation time and refractivity for various values of refractivity is depicted in Figure 13. The figure is read by selecting the propagation time on the horizontal axis, tracing up to the appropriate refractivity line, and reading the range bias off of the vertical axis. The range bias associated with refractivity mismatch, given propagation time, may be read as the vertical difference in range bias between two curves.

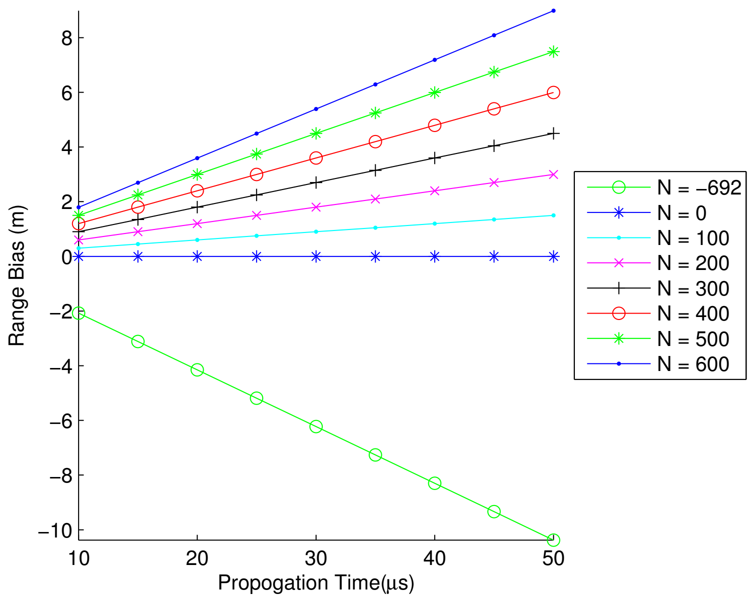

Figure 13. Range bias as a function of propagation time given refractivity and using the true value of the speed of light.

The range bias in meters per ten microseconds of propagation time under three different total atmospheric pressure scenarios as a function of both ambient temperature and dew-point spread is shown in Figures 14 through 16. Note that dew-point temperature is ambient temperature minus the dew-point spread. We can clearly see from these plots that both dry areas (high dew-point spread) and cold areas have less impact on range estimates than wet areas (low dew-point spread) and high temperatures. Likewise, under all temperature and dew-point spread conditions, higher total pressure conditions have more impact than lower total pressure conditions.

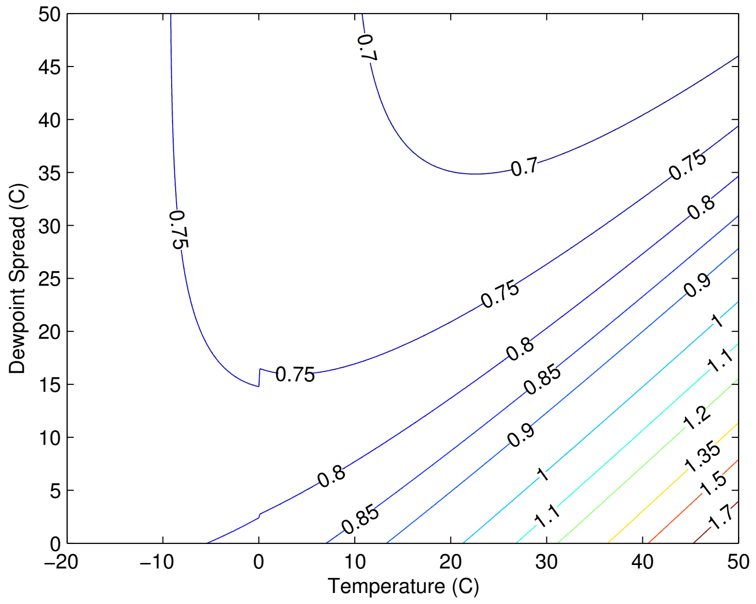

Figure 14. Range bias per 10μ seconds of propagation time at 850 millibar total atmospheric pressure and using the true value of the speed of light.

Another way to view the range bias is to normalize it by some other quantity of interest. One such quantity is the standard deviation of the range estimate error $\sigma\{\epsilon_\tau\} = c\tau/\sqrt{12}$, where ϵ_τ is a uniform random variable over the interval $[0, c\tau]$. The ratio of the range bias per ten microseconds normalized by $\sigma\{\epsilon_\tau\}$ is

$$f(n_0, \hat{n}, \tau) = \frac{b_d}{\sigma\{\epsilon_\tau\}} = \frac{10\mu\sqrt{12}}{\tau}\frac{\hat{n} - n_0}{\hat{n}}. \tag{49}$$

When this ratio is small, the range bias is overwhelmed by the error associated with using timing adjust units to measure distance. Likewise, when the ratio is larger, it suggests a performance gain may be obtained by incorporating refractivity explicitly. It is notable that, had the timing adjust unit error structure been Gaussian instead of uniform, this would

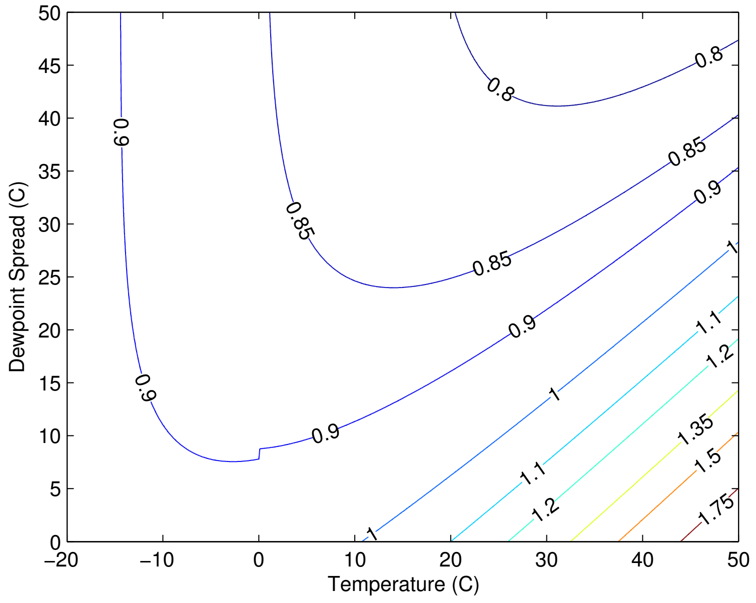

Figure 15. Range bias per 10µ seconds of propagation time at 1000 millibar total atmospheric pressure and using the true value of the speed of light.

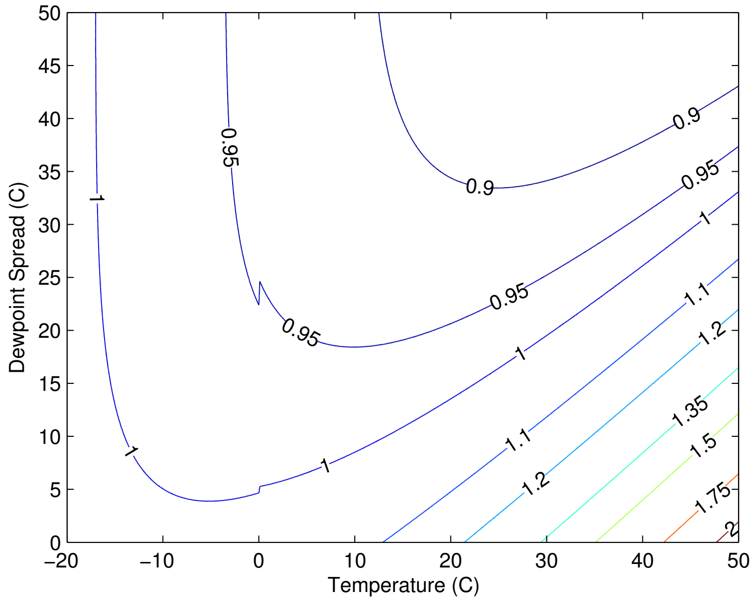

Figure 16. Range bias per 10µ seconds of propagation time at 1100 millibar total atmospheric pressure and using the true value of the speed of light.

be a Z-test of the hypothesis that the refractivity bias is different from zero in the presence of timing noise. The logic of a hypothesis test still holds even if the critical values do not. Further, the usual significance thresholds do not apply because the refractivity bias is fed forward through the position or velocity estimator. Therefore, it is anticipated that a much lower value than would otherwise be accepted as significant will result in meaningful change in the final position or velocity estimate error structure.

A plot of $b_d/\sigma\{\epsilon_\tau\}$ for a range of timing adjust values in seconds is shown in Figure 17. There are three vertical lines demarcating timing adjust units associated with three different system bandwidths. A baseline refractivity of 350 ppm was chosen as a proxy for a “normal” value of refractivity in the atmosphere. At relatively low system bandwidths (below 10 MHz) there is little reason to believe refractivity makes a noticeable difference regardless of the assumed refractivity. Even at relatively high system bandwidths (50 MHz or more) there is a fairly large tolerable refractivity estimate error before the test statistic might become large enough to cause concern.

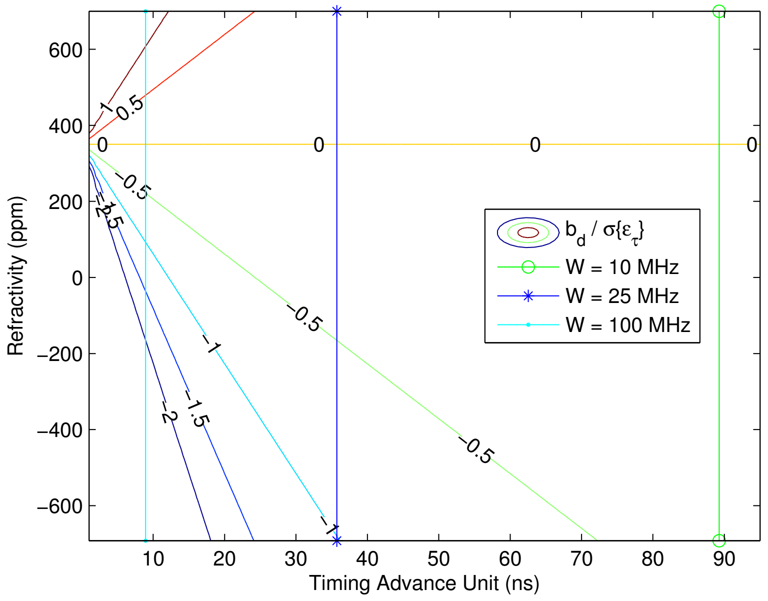

Figure 17. Range bias normalized by timing adjust unit-based range error standard deviation.

C. TARGET VELOCITY BIAS

A method was presented in Chapter II.C for estimating true receiver velocity from observed frequency shift. As in the case of distance estimates discussed in Section B, these estimates are also impacted by failure to account for refractivity. An expression for this bias is derived in this section.

To compute the bias due to refractivity mismatch, treat v_p as the estimated propagation velocity and substitute

$$v_p = \frac{c}{\hat{n}} + \frac{c}{n_0} - \frac{c}{n_0} \tag{50}$$

into (36). This substitution yields

$$\mathbf{v} = \frac{\left(\frac{c}{n_0} + \frac{c(n_0 - \hat{n})}{\hat{n} n_0}\right)^2 \mathbf{R}_{j,i}^{-1} \mathbf{f}_{j,i}}{f_t}. \tag{51}$$

Like (47), (51) is unbiased when $n_0 = \hat{n}$. Expand the squared term in (51) to obtain

$$\mathbf{v} = \frac{\left(\frac{c^2}{n_0^2} + \frac{c^2(n_0 - \hat{n})^2}{\hat{n}^2 n_0^2} + 2\frac{c^2(n_0 - \hat{n})}{n_0^2 \hat{n}}\right) \mathbf{R}_{j,i}^{-1} \mathbf{f}_{j,i}}{f_t}. \tag{52}$$

By inspection of (52), the velocity bias may now be expressed as

$$\mathbf{b}_v(n_0, \hat{n}) = \frac{\left(\frac{c^2(n_0 - \hat{n})^2}{\hat{n}^2 n_0^2} + 2\frac{c^2(n_0 - \hat{n})}{n_0^2 \hat{n}}\right) \mathbf{R}_{j,i}^{-1} \mathbf{f}_{j,i}}{f_t}. \tag{53}$$

Noting $(n_0 - \hat{n})^2$ is on the order of 10^{-8}, we see that the first term of the bias is approximately four orders of magnitude smaller than the second. Thus, a simplified estimate of the velocity bias is given by

$$\hat{\mathbf{b}}_v(n_0, \hat{n}) \approx 2\frac{c^2(n_0 - \hat{n})}{f_t n_0^2 \hat{n}} \mathbf{R}_{j,i}^{-1} \mathbf{f}_{j,i}. \tag{54}$$

In summary, three directly characterizable biases arising from refractivity mismatch were presented in this chapter. These are propagation velocity bias, range estimate bias, and target velocity bias. Equations for each were presented. Propagation velocity bias feeds forward into each of the other two biases. Range estimate bias is a function of local atmospheric parameters. It may or may not be significant in relation to the timing noise depending on the bandwidth of the system in question. Finally, it is possible to use a simple estimate for target velocity error by recognizing a single component of the full estimator will tend to dominate the result.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RECEIVER PAIR SELECTION

The methods presented in Chapter II each culminate in solving a matrix equation. As a result, performance of the algorithm is tied to this matrix equation being well-conditioned. An algorithm for choosing a set of receivers which may be expected (but not guaranteed) to be well conditioned in the context of either the the TOA, TDOA, or DVE problem is presented in this chapter. This is done by application of graph theory and elementary linear algebra.

An example of a situation in which tower choice is possible is shown in Figure 18. In this example, a single transmitter is observed (lightning bolt) at seven geographically

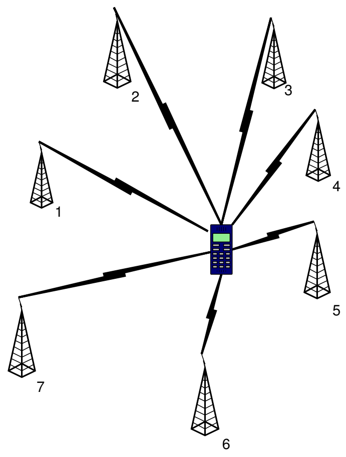

Figure 18. An example of tower choice in which seven receivers are available of which four or five are required.

dispersed receivers. As only three to five of these are needed (depending on the algorithm employed) to construct an estimate, it is possible to choose from amoungst them the “best” possible subset. For example, if a TDOA solution is desired, then towers $1, 2, 3, 4, 5$ or $2, 4, 5, 6, 7$ might be used.

This chapter is organized as follows. The shared terms and concepts are presented in Section A. The application of these to the TOA problem are presented in Section B.

The application of the shared terms and concepts to the TDOA problem is presented in Section C. Finally, the application to the DVE problem is presented in Section D.

A. TERMINOLOGY

The material which is common to formulating the methods presented in the later sections of this chapter is presented in this section. First is a brief listing of the graph theory terminology used herein. Second is a high level discussion of the proposed method to select linear constraints from the overall set thereof.

1. Notation

A graph G is comprised of a set of vertexes $V(G)$ and a set of edges $E(G)$. These are referred to as V and E where no confusion results. An edge connects two vertexes. The order of a graph $|V|$ is the number of vertexes and the size of a graph $|E|$ is the number of edges. Denote any vertex as $v_i \in V$ such that $i = 1, 2, \ldots, |V|$ and any edge as $v_i v_j \in E$ such that $i, j \in \{i = 1, 2, \ldots, |V|\}$. A subgraph of G is any graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. All definitions are taken from [44].

With respect to the constraints, there is also a definition common to all three proposed methods.

Definition A.1 (Admissable Constraint)**.** *A constraint is said to be admissible if and only if there is not an a priori reason to exclude the constraint from any possible solution.*

2. Choosing Linear Constraints

Suppose, in the problem at hand, it is possible to form a set of linear constraints greater than the number necessary for solution. In the methods given in Chapter II, this would mean being able to choose any three from some larger set of possible constraints. These equations might then be arranged into the matrix equation

$$\underset{3\times 3}{\mathbf{A}}\, x = \underset{3\times 1}{\mathbf{b}} \tag{55}$$

where x is the quantity of interest. For the system of equations to be well-conditioned, the eigenvalues of the $\mathbf{A}$ matrix of (55) must have a sufficiently large minimum value. Following [45], we let $\mathbf{V}_i$ be the vector which forms the row i of $\mathbf{A}$. Consider

$$|\mathbf{A}| = v_1 v_2 v_3 \cos\phi \sin\theta \tag{56}$$

where $v_i = |\mathbf{V}_i|$, θ is the angle between $\mathbf{V_2}$ and $\mathbf{V}_3$ and ϕ is the angle between $\mathbf{V}_1$ and the normal to the plane of $\mathbf{V}_2$ and $\mathbf{V}_3$ [45, pg. 325]. While a comprehensive search for the best possible set of constraint equations may be prohibitively expensive, a reasonable approach is to first identify a set of independent constraints and then apply the fact given in (56) to pick a set of three which either yield a large eigenvalue or are as reasonably close to orthogonal as may be possible. This suggests the algorithm shown in Fig. 19. The former approach is chosen in this thesis.

```
Require: A description of the set of possible linear constraints.
Ensure: Constraint matrix $\mathbf{A}$ eigenvalues are bounded from below in absolute value.
 1: Compute all possible rows of $\mathbf{A}$.
 2: Select any acceptable row $\mathbf{V}_3$.
 3: Select $\mathbf{V}_2$ maximizing $|\mathbf{V}_3 \times \mathbf{V}_2|$.
 4: Select $\mathbf{V}_1$ maximizing $|\mathbf{V}_1 \cdot (\mathbf{V}_3 \times \mathbf{V}_2)|$.
```

Figure 19. The proposed constraint selection algorithm.

B. APPLICATION TO CIRCULAR MULTILATERATION

A proposed method for using graph theory and the linear constraint choice algorithm presented in Figure 19 to the problem of circular multilateration is presented in this section. In this section, the proposed method is presented and the computational complexity of this algorithm is derived.

1. Receiver Pair Choice Algorithm

In order to translate the TOA geolocation problem into graph theoretic terms, it is necessary to identify the vertexes and edges of the graph. This model is referred to as the TOA constraint graph.

Definition B.1 (TOA Constraint Graph)**.** *Given a set of receivers and a set of propagation time measurements, the TOA constraint graph* G *is defined with a vertex set* $V(G)$ *equal to the set of admissible observers and an edge set* $E(G)$ *whose members are the admissible observer pairings where each pairing represents a circular constraint.*

By convention, the TOA constraint graph do not contain as vertexes receivers that did not receive the signal of interest or measure a negative propagation time. It is important that observers with negligible propagation time be excluded because the circular constraint is degenerate. Furthermore, it is assumed that none of the degenerate circumstances discussed in Chapter II are present.

It is now necessary to prove that sets of independent circular constraints are equivalent to trees defined on the TOA constraint graph in Definition B.1.

Theorem B.1. *Given a TOA constraint graph* G*, any connected subgraph* H *of* G *is a tree if and only if the edge set of* H *represents a set of independent circular constraints.*

Proof. ($\Leftarrow$)Proceed by proving the contra-positive. Suppose H is a subgraph of G, which is not a tree. Then it must contain a cycle [44, pg. 83]. Any cycle, written as a sequence of vertexes, has form

$$p_{\alpha_1} \to p_{\alpha_2} \to \ldots \to p_{\alpha_n} \to p_{\alpha_1} \tag{57}$$

where α is the ordered set of indices of the observers included in the cycle. Now invoke Definition B.1 to write the set of edges represented in (57) as circular constraints in the form of the intersection of two spheres

$$\frac{|p_0 - p_{\alpha_2}|}{r_{\alpha_2}} - \frac{|p_0 - p_{\alpha_1}|}{r_{\alpha_1}} = 0 \tag{58}$$

$$\frac{|p_0 - p_{\alpha_3}|}{r_{\alpha_3}} - \frac{|p_0 - p_{\alpha_2}|}{r_{\alpha_2}} = 0 \tag{59}$$

$$\vdots$$

$$\frac{|p_0 - p_{\alpha_n}|}{r_{\alpha_n}} - \frac{|p_0 - p_{\alpha_{n-1}}|}{r_{\alpha_{n-1}}} = 0 \tag{60}$$

$$\frac{|p_0 - p_{\alpha_1}|}{r_{\alpha_1}} - \frac{|p_0 - p_{\alpha_n}|}{r_{\alpha_n}} = 0. \tag{61}$$

Adding (58) through (60), we get

$$\frac{|p_0 - p_{\alpha_n}|}{r_{\alpha_n}} - \frac{|p_0 - p_{\alpha_1}|}{r_{\alpha_1}} = 0. \tag{62}$$

As (62) is equal to (61), the latter is dependent upon the remainder.

($\Rightarrow$)Now suppose H is a tree. By the definition of the TOA constraint graph, each edge represents a circular constraint. Let the edge set of H be denoted by β whose elements are the un-ordered pairs of vertexes forming the respective edges. Denote the ith edge of H as β_i where $\beta_{i,j}$, $j \in \{1, 2\}$ denotes one vertex of the pair. From the definition of the TOA constraint graph, each β_i is a circular constraint. As H is a tree with N vertexes, it has $N - 1$ edges. These constraints may be expressed in the form of intersections of spheres as

$$\frac{|p_0 - p_{\beta_{1,2}}|}{r_{\beta_{1,2}}} - \frac{|p_0 - p_{\beta_{1,1}}|}{r_{\beta_{1,1}}} = 0 \tag{63}$$

$$\frac{|p_0 - p_{\beta_{2,2}}|}{r_{\beta_{2,2}}} - \frac{|p_0 - p_{\beta_{2,1}}|}{r_{\beta_{2,1}}} = 0 \tag{64}$$

$$\vdots$$

$$\frac{|p_0 - p_{\beta_{N-1,2}}|}{r_{\beta_{N-1,2}}} - \frac{|p_0 - p_{\beta_{N-1,1}}|}{r_{\beta_{N-1,1}}} = 0. \tag{65}$$

Without loss of generality, it may be assumed that each sphere has a unique center as by construction the distance $r_i = r_j$ whenever $p_i = p_j$. Equations (63)–(65) form a system of $N - 1$ equations linear in N distinct spheres; therefore, the constraints must be independent. $\square$

This result allows graph theoretic algorithms to be employed in identifying sets of independent constraint equations from which a final geolocation solution may be constructed. The algorithm shown in Figure 20 is proposed.

```
Require: a tree $H \subseteq G$ such that $G$ conforms to Definition B.1.
Ensure: Constraint matrix $\mathbf{A}$ eigenvalues are bounded from below.
  1: Compute all possible rows of $\mathbf{A}$ with form given by the left hand side of (20), $\vec{n}_{ij}$.
  2: Select any acceptable row $\mathbf{V}_3$.
  3: Select $\mathbf{V}_2$ maximizing $|\mathbf{V}_3 \times \mathbf{V}_2|$.
  4: Select $\mathbf{V}_1$ maximizing $|\mathbf{V}_1 \cdot (\mathbf{V}_3 \times \mathbf{V}_2)|$.
```

Figure 20. The proposed constraint selection algorithm for TOA position estimation.

It is possible to incorporate *a priori* information about the prefer-ability of various edges of the TOA constraint graph though the use of some rule for assigning weights to said edges. One such rule when used with a maximum weight spanning tree algorithm is given by

$$w_c = \frac{\max\{d_i, d_j\}}{|p_j - p_i|}. \tag{66}$$

If a minimum weight spanning tree algorithm is used, then exchanging $\min\{d_i, d_j\}$ for $\max\{d_i, d_j\}$ produces an appropriate weighting scheme. This weight function by construction prefers circular constraints with small radii. It is assumed that any such constraints where the radius is too small have been eliminated from consideration, thereby, preventing the weight function from giving preference to those constraints that are unacceptable for other reasons.

2. Computational Complexity

It is also possible to determine the computational complexity of this modified algorithm. The case of a single target is considered because extension to multiple targets is elementary. Let n be the number of receivers to be considered. Then there are

$$n_c = \binom{n}{2} = \frac{n(n-1)}{2} \approx 0.5n^2 \tag{67}$$

possible constraints from which to choose an independent subset. This choice retains $n-1$ circular constraints. Via the direct mapping to linear constraint equations, there are

$$n_p = \binom{n-1}{2} = \frac{(n-1)(n-2)}{2} \approx 0.5n^2 \tag{68}$$

pairs of linear constraints to check in order to find the most orthogonal pair. Once this pair is identified, there are $n-3$ further checks to perform to find the final constraint. This leads to a total complexity of

$$n_c + n_p + n - 3 \approx n^2 + n \mapsto O(n^2). \tag{69}$$

C. APPLICATION TO HYPERBOLIC MULTILATERATION

A proposed method for the application of tower choice to the problem of TDOA geolocation is presented in this section. There are three major components. The first is a modified version of the algorithm given in [13] to enable receiver choice. Second is the proposed method for using the generalized linear constraint equation and the concepts presented in Section A to formulate the proposed constraint choice algorithm. Third is a presentation of an analysis of the computational complexity of the proposed method.

1. Modified TDOA Position Estimation Algorithm

This work uses a modified version of the solution given in [13]. Reformulation is necessary for two reasons. First, the refractivity situation considered here is a special case of that considered in [13] ergo simplification is possible. Second, the original assumes that a single receiver is used as the reference receiver. This assumption yields a much nicer set of final equations but greatly restricts the ways in which receiver choice may be employed.

Assuming that the refractivity of the local medium is constant, we get the hyperbolic constraint

$$\frac{|p_2|^2}{n^{-2}} - \frac{|p_1|^2}{n^{-2}} - 2\left(\frac{p_2^T}{n^{-2}} - \frac{p_1^T}{n^{-2}}\right) p_0 = c^2(t_2^2 - t_1^2) - 2t_0c^2(t_2 - t_1), \qquad (70)$$

where the α_2 and α_1 of the original are replaced with n^{-1}, is exact [13, (9)]. The position of the transmitter to be located is $\vec{p}_0$, the location of each observing receiver is $\vec{p}_i$ where $i \in \mathbb{N}$, t_0 is the time of transmission, and t_i where $i \in \mathbb{N}$ is the time of reception at receiver i. Equation (70) is a special case of

$$|p_j|^2 - |p_i|^2 - 2\left(p_j^T - p_i^T\right) p_0 = c_n^2(t_j^2 - t_i^2) - 2t_0c_n^2(t_j - t_i). \qquad (71)$$

Letting k and l denote another pair of receivers, we solve (71) for $2t_0c_n^2$:

$$\frac{-1}{(t_l - t_k)}\left[|p_l|^2 - |p_k|^2 - 2\left(p_l^T - p_k^T\right) p_0 - c_n^2(t_l^2 - t_k^2)\right] = 2t_0c_n^2. \qquad (72)$$

Combine (71) and (72) to form the constraint equation

$$\begin{aligned} &|p_j|^2 - |p_i|^2 - 2\left(p_j^T - p_i^T\right) p_0 - c_n^2(t_j^2 - t_i^2) \\ &\qquad = \frac{(t_j - t_i)}{(t_l - t_k)}\left[|p_l|^2 - |p_k|^2 - 2\left(p_l^T - p_k^T\right) p_0 - c_n^2(t_l^2 - t_k^2)\right]. \qquad (73) \end{aligned}$$

Rearrange (73) to obtain

$$\begin{aligned} &2\left[\frac{(t_j - t_i)}{(t_l - t_k)}\left(p_l^T - p_k^T\right) - \left(p_j^T - p_i^T\right)\right] p_0 \\ &\qquad = \frac{(t_j - t_i)}{(t_l - t_k)}\left(|p_l|^2 - |p_k|^2 - c_n^2(t_l^2 - t_k^2)\right) - |p_j|^2 + |p_i|^2 + c_n^2(t_j^2 - t_i^2). \qquad (74) \end{aligned}$$

Following [13], we can form three linearly independent constraints using five receivers. These constraints may be written in the form of the matrix equation

$$\underset{3\times 3}{\mathbf{A}} p_0 = \underset{3\times 1}{\mathbf{b}} \tag{75}$$

where each row of **A** is given by the left hand side of (74) for some choice of $i, j, k,$ and l and the respective entry of **b** is given by the right hand side of (74). Assuming that the receiver pairs have been chosen to ensure linear independence, we can solve (75) directly.

This new version is susceptible to degeneracy under certain geometric configurations of the observing receivers. The first is if all five receivers are coplaner, in which case the final system is under-determined for three dimensional localization. The second is if any four of the receivers are colinear as the fourth receiver yields no additional information. This is because when the two hyperbola formed using the first three receivers are intersected, a circular constraint in formed. Imagine "sliding" one of the first three receivers along the common line until it is in the position of the fourth receiver. Since the target emitter is a constant distance from the common line, the intersecting circle never changes; therefore, the fourth receiver contributes nothing.

2. Receiver Pair Choice Algorithm

In order to translate the TDOA geolocation problem into graph theoretic terms, it is necessary to identify the vertexes and edges of the graph. This model is referred to as the TDOA constraint graph.

Definition C.1 (TDOA Constraint Graph)**.** *Given a set of observers and a set of time-of-arrival measurements, the TDOA constraint graph* G *is defined with a vertex set* $V(G)$ *equal to the set of observers and an edge set* $E(G)$ *whose members are the admissible observer pairings where each pairing represents a hyperbolic constraint.*

By convention, the TDOA constraint graph does not contain as vertexes observers that do not receive the signal of interest. One possible reason to exclude a constraint and its associated edge is negligible time difference of arrival between the two observers in question (which may give rise to computational degeneracy in some algorithms). It is further assumed that the degenerate geometries described in the last section are not present.

To establish the relevance of the graph theoretic model of the constraint space, it is sufficient to prove that, in the context of TDOA algorithms, any subgraph of the TDOA constraint graph, which is a tree, has edges that form a linearly independent set of hyperbolic constraints.

Theorem C.1. *Given a TDOA constraint graph* G*, any connected subgraph* H *of* G *is a tree if and only if the edge set of* H *represents a set of independent hyperbolic constraints.*

Proof. ($\Leftarrow$)Proceed by proving the contra-positive. Suppose H is a subgraph of G, which is not a tree. Then it must contain a cycle [44, pg. 83]. Any cycle, written as a sequence of vertexes, has form

$$p_{\alpha_1} \to p_{\alpha_2} \to \ldots \to p_{\alpha_n} \to p_{\alpha_1} \tag{76}$$

where α is the ordered set of indices of the observers included in the cycle. Now invoke Definition C.1 to write the set of edges represented in (76) as hyperbolic constraints:

$$D^2_{\alpha_2} - D^2_{\alpha_1} = \nu_p \left(t^2_{\alpha_2} - t^2_{\alpha_1}\right) \tag{77}$$

$$D^2_{\alpha_3} - D^2_{\alpha_2} = \nu_p \left(t^2_{\alpha_3} - t^2_{\alpha_2}\right) \tag{78}$$

$$\vdots$$

$$D^2_{\alpha_N} - D^2_{\alpha_{N-1}} = \nu_p \left(t^2_{\alpha_N} - t^2_{\alpha_{N-1}}\right) \tag{79}$$

$$D^2_{\alpha_1} - D^2_{\alpha_N} = \nu_p \left(t^2_{\alpha_1} - t^2_{\alpha_N}\right). \tag{80}$$

Adding (77)–(79), we get

$$D^2_{\alpha_N} - D^2_{\alpha_1} = \nu_p \left(t^2_{\alpha_N} - t^2_{\alpha_1}\right). \tag{81}$$

Equation (81) is equal to (80), ergo the set of constraints is dependent.

($\Rightarrow$)Now suppose H is a tree. Then by the definition of the TDOA constraint graph, each edge represents a hyperbolic constraint. Let the edge set of H be denoted by β whose elements are the un-ordered pairs of vertexes that form the respective edges. Denote the ith edge of H as β_i where $\beta_{i,j}$, $j \in \{1, 2\}$ denotes one vertex of the pair. From the definition of the TDOA constraint graph, each β_i is a hyperbolic constraint. These constraints may be expressed as

$$\begin{aligned}
D^2_{\beta_{1,2}} - D^2_{\beta_{1,1}} &= \nu_p \left(t^2_{\beta_{1,2}} - t^2_{\beta_{1,1}}\right) \\
D^2_{\beta_{2,2}} - D^2_{\beta_{2,1}} &= \nu_p \left(t^2_{\beta_{2,2}} - t^2_{\beta_{2,1}}\right) \\
&\vdots \\
D^2_{\beta_{N-2,2}} - D^2_{\beta_{N-2,1}} &= \nu_p \left(t^2_{\beta_{N-2,2}} - t^2_{\beta_{N-2,1}}\right) \\
D^2_{\beta_{N-1,2}} - D^2_{\beta_{N-1,1}} &= \nu_p \left(t^2_{\beta_{N-1,2}} - t^2_{\beta_{N-1,1}}\right).
\end{aligned} \tag{82}$$

As H is a tree with N vertexes, then it has $N-1$ edges and there are $N-1$ constraint equations [44, Theorem 4.2]. As (82) is a linear system in $\{D^2_{\beta_{j,i}} | i = 1, 2, \ldots, N-1; j = 1, 2\} \equiv \{D^2_i | i = 1, 2, \ldots, N\}$ which is under-determined, the equations must be independent. This concludes the proof. $\square$

The link between the graph theoretic model and the usual TDOA constraint problem is provided by Theorem C.1. Sets of independent hyperbolic constraints may be identified by searching the TDOA graph for trees of a specified minimum size. The proposed algorithm for performing said search is shown in Fig. 21.

Additional refinement is possible by incorporating information about which hyperbolic constraints are *a priori* preferable to others. It should be noted that in cases where there is negligible difference in the time-of-arrival between two observing receivers, viz. $|t_j - t_i| \approx 0$, the estimation method given in Section C.1 becomes numerically unstable. More generally, this is the situation in which the hyperbola approximates a plane. In order to avoid constraints where this is true,

$$w_h(i, j) = \frac{|t_j - t_i| c_n}{|p_j - p_i|} \tag{83}$$

```
**Require:** a tree $H \subseteq G$ such that G conforms to Definition C.1.
**Ensure:** Constraint matrix **A** eigenvalues are bounded from below in absolute value.
1: Compute all possible rows of **A** with form given by
       $\frac{(t_j - t_i)}{(t_l - t_k)}\left(\vec{p}_l^T - \vec{p}_k^T\right) - \vec{p}_j^T + \vec{p}_i^T,$
   the from the left hand side of (74).
2: Select any acceptable row $\mathbf{V}_3$.
3: Select $\mathbf{V}_2$ maximizing $|\mathbf{V}_3 \times \mathbf{V}_2|$.
4: Select $\mathbf{V}_1$ maximizing $|\mathbf{V}_1 \cdot (\mathbf{V}_3 \times \mathbf{V}_2)|$.
```

Figure 21. The proposed constraint selection algorithm for TDOA position estimation.

is one possible rule for assigning weights to the edges of the TDOA constraint graph. This rule recognizes the fact that when the distance $|p_j - p_i|$ between receivers i and j is very large, correspondingly larger time differences are required to have the constraint surface to be significantly different from a plane in the near field. This rule will work with a maximum weight spanning tree algorithm. If a minimum weight spanning tree algorithm is used, then the reciprocal of this weight function will serve.

3. Computational Complexity

It is also possible to compute the computational complexity of this modified algorithm. The analysis is undertaken for a simple target as extending the result to multiple targets is elementary.

Let n denote the number of receivers to be considered in the analysis. Then the number of possible receiver pairs (hyperbolic constraints) is given by

$$n_h = \binom{n}{2} = \frac{n(n-1)}{2} \approx 0.5n^2. \tag{84}$$

This is also the total number of weights that must be computed if a weighting function is used. Of these n_h possible hyperbolic constraints, the spanning tree algorithm retains $n-1$

of them. As each linear constraint requires two hyperbolic constraints, there are

$$n_c = \binom{n-1}{2} = \frac{(n-1)(n-2)}{2} \approx 0.5n^2 \tag{85}$$

such constraints from which to choose. The process of choosing constraints first requires the most orthogonal pair of linear constraints be identified. There are

$$n_p = \binom{n_c}{2} = \frac{n_c(n_c-1)}{2} \approx 0.5n_c^2 \approx 0.125\left(n^2\right)^2 = 0.125n^4 \tag{86}$$

such pairs to check. Finally, once the most orthogonal pair has been identified, its normal vector must be checked against $n_c - 2$ other linear constraints to find the third linear constraint equation. This leads to an approximate total number of operations on the order of

$$n_h + n_c + n_p + n_c \approx 0.125n^4 + n^2 \mapsto O(n^4). \tag{87}$$

D. APPLICATION TO DOPPLER VELOCITY ESTIMATION

As the Doppler velocity estimator presented in Chapter II is already in the form of (55), application of the linear constraint choice algorithms is straightforward. The proposed algorithm is shown in Fig. 22.

```
Require: A set of receivers which receive a Doppler shifted transmission.
Ensure: Constraint matrix $\mathbf{R}$ eigenvalues are bounded from below.
1: Compute all possible rows of $\mathbf{A}$ with form given by the left hand side of (35), $\frac{\vec{r}_j}{t_j} - \frac{\vec{r}_i}{t_i}$ such that $i \neq j$.
2: Select any acceptable row $\mathbf{V}_3$.
3: Select $\mathbf{V}_2$ maximizing $|\mathbf{V}_3 \times \mathbf{V}_2|$.
4: Select $\mathbf{V}_1$ maximizing $|\mathbf{V}_1 \cdot (\mathbf{V}_3 \times \mathbf{V}_2)|$.
```

Figure 22. The proposed constraint selection algorithm for Doppler velocity estimation.

The following definition may be used to enable use of graph theoretic approaches to inform the constraint choice problem.

Definition D.1 (Doppler Constraint Graph)**.** *Given a set of observers and a set of received frequency measurements, the Doppler constraint graph* G *is defined with a vertex set* $V(G)$ *equal to the set of observers and an edge set* $E(G)$ *whose members are the admissible observer pairings where each pairing represents a linear constraint.*

As the constraint equations used in the Doppler velocity estimator are linear, any set of four or more is linearly dependent. Therefore, graph theory cannot be used to select sets of independent constraints as before. It is possible, however, to use graph theory to keep a subset of constraints incorporating all observers and may, based on *a priori* reasoning, be the most fruitful subset to which to apply the algorithm shown in Figure 22.

One possible way to choose a preferred subset of possible constraint equations incorporating all observers is to assign weights to the various edges of the Doppler constraint graph and then apply a weight-optimized spanning tree algorithm. In the context of Doppler velocity estimation, two kinds of constraint equations may be preferable to others. The first kind has a very high absolute frequency shift $|f_{j,i}|$, which indicates that the motion of the target at the time of the observation is mostly in the line of sight between receivers i and j. The second kind has an absolute frequency shift $|f_{j,i}|$ very close to zero, which indicates the motion of the target at the time of observation is very close to perpendicular to the line of sight between receivers i and j. The useful consequence of using such a rule in assigning weights is that the constraint with the highest and lowest absolute frequency shift are by construction most orthogonal and, therefore, good candidates to be selected by the constraint choice algorithm. Methods for the assignment of such weights are beyond the scope of this work.

Proposed methods for implementing receiver choice for TOA, TDOA, and DVE were presented in this chapter. In all cases, this requires some understanding of the relation between the structure of the rows of a square matrix and its determinant as well as some rudimentary graph theory. While the method may be applied in all three contexts, the computational complexity varies widely. For the TOA and DVE methods, the complexity is $O(n^2)$ in the number of receivers n whereas it is $O(n^4)$ for the TDOA algorithm.

V. SIMULATION AND RESULTS

A simulation study was undertaken to provide proof of concept and to assess impact of application of the techniques presented or developed in Chapters II, III, and IV to methods developed in Chapter IV in an IEEE 802.16 compliant OFDMA network. This is facilitated by the development of the **Geolocation** package for MATLAB, which contains all the necessary functionality to carry out the work undertaken in this chapter.

An overview of the process of simulating the geolocation process is given in Section A. The class structure of the simulation software is presented in Section B. Details of how the various simulations were designed are given in Section C. The result of the simulations undertaken are given in Section D. A discussion of the results is presented in Section E. Many of the diagrams presented in this chapter are based on the Universal Modeling Language (UML) 2.0 standard as described in [46]. See Appendix A for an introduction to the UML diagrams used in this chapter as well as the modifications to the standard used in this thesis. The class definitions are to be found in Appendix B. All header and inline comments have been retained in the appendix for the interested reader.

When an algorithm is under discussion, it is denoted in a normal font. When the software implementing said algorithm is under discussion, its typeface matches the UML diagram notation. All classes are in boldface and methods are in a Courier font.

A. SIMULATION PROCESS

The process of simulating the geolocation problem can be approached from the perspective of the activities involved in running a single scenario. A UML 2.0 activity diagram of this process is shown in Figure 23.

The activity diagram is divided into three major subdivisions. These are initialization, simulation, and analysis. The environment, network, and target must be created to initialize the scenario. The three activities are placed between fork and join bars in the

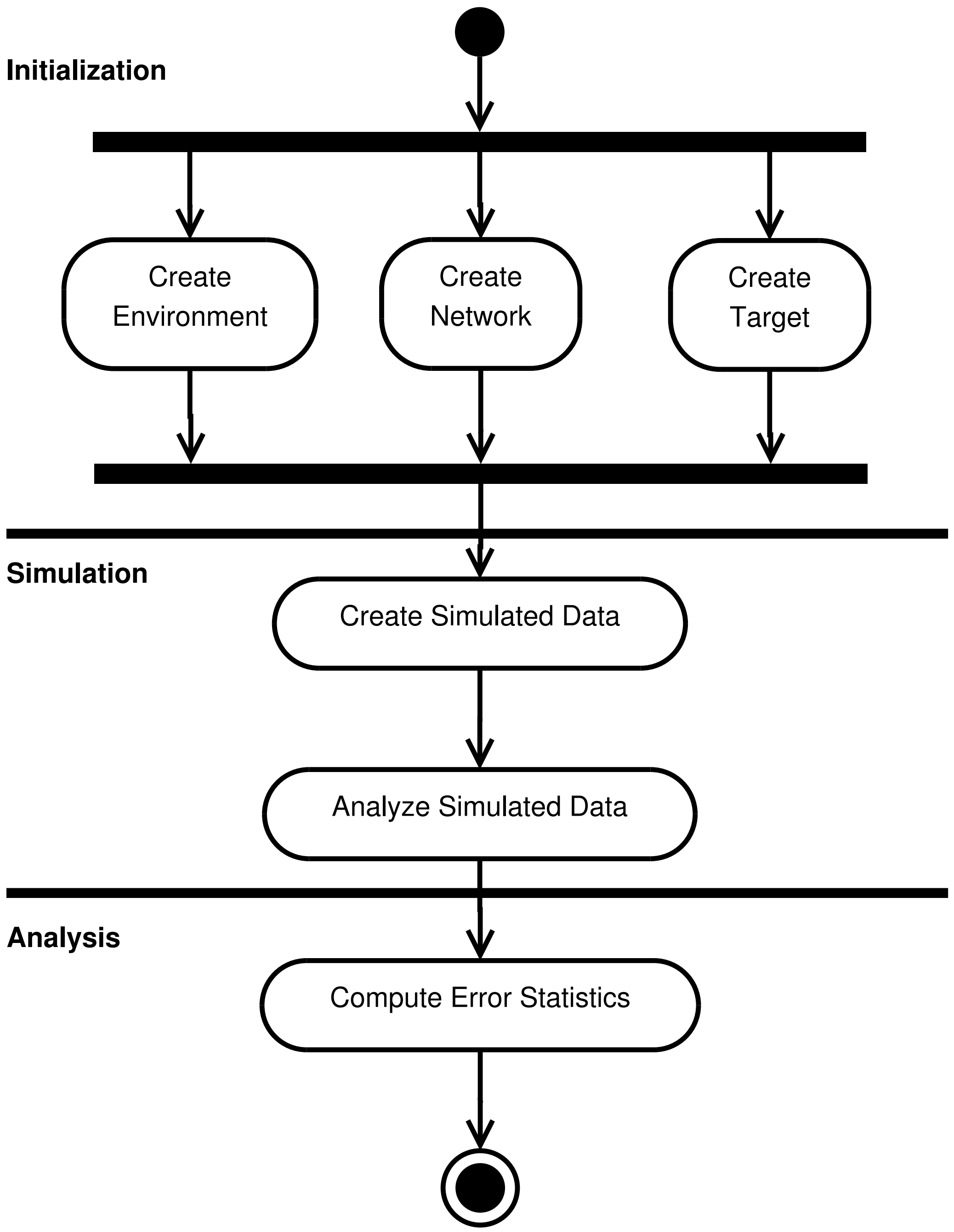

Figure 23. Activity diagram for a single simulation scenario.

activity diagram because they are mutually independent and, therefore, may be done in any order, if not actually in parallel. The environment and network are assumed to be static throughout the scenario. The target may either be a single target or a collection of targets as needed. Once the scenario has been initialized, simulation may begin. Simulated timing adjust (TA) and frequency adjust (FA) data is computed for each target observation in the network expressed in IEEE 802.16 OFDMA-compliant units as described in Chapter II. Next, the simulated data may be analyzed using any or all of the methods presented in Chapters II through IV. Finally, these solutions are then compared to the original target information in order to compute the error statistics needed for further analysis of algorithm performance.

Due to the inherently modular nature of the process, it is interesting to consider how the activity diagram changes if multiple simulation runs are considered. In this case, it is necessary to add a several optional branches that control the flow through various blocks, which may be optionally reset. It is assumed that the analysis method and statistics collected will be common across runs. These changes are reflected in the activity diagram presented in Figure 24.

B. CLASS STRUCTURE OF THE SIMULATION SOFTWARE

In order to carry out the simulations, an object-oriented MATLAB package was developed. This package, called **Geolocation**, has a high-level structure as represented in the diagram shown in Figure 25. This structure consists of five major parts: the **Environment**, **Network**, **Target**, and **Data** classes with their associated subclasses and the **Analysis** subpackage with its constituent classes. This section does not include a discussion of the various stand-alone utility functions that do not belong to any class but are included as part of the full **Geolocation** package. Each of the groups of classes and their inter-relationships will be discussed in the relevant section below as well as any special implementation details.

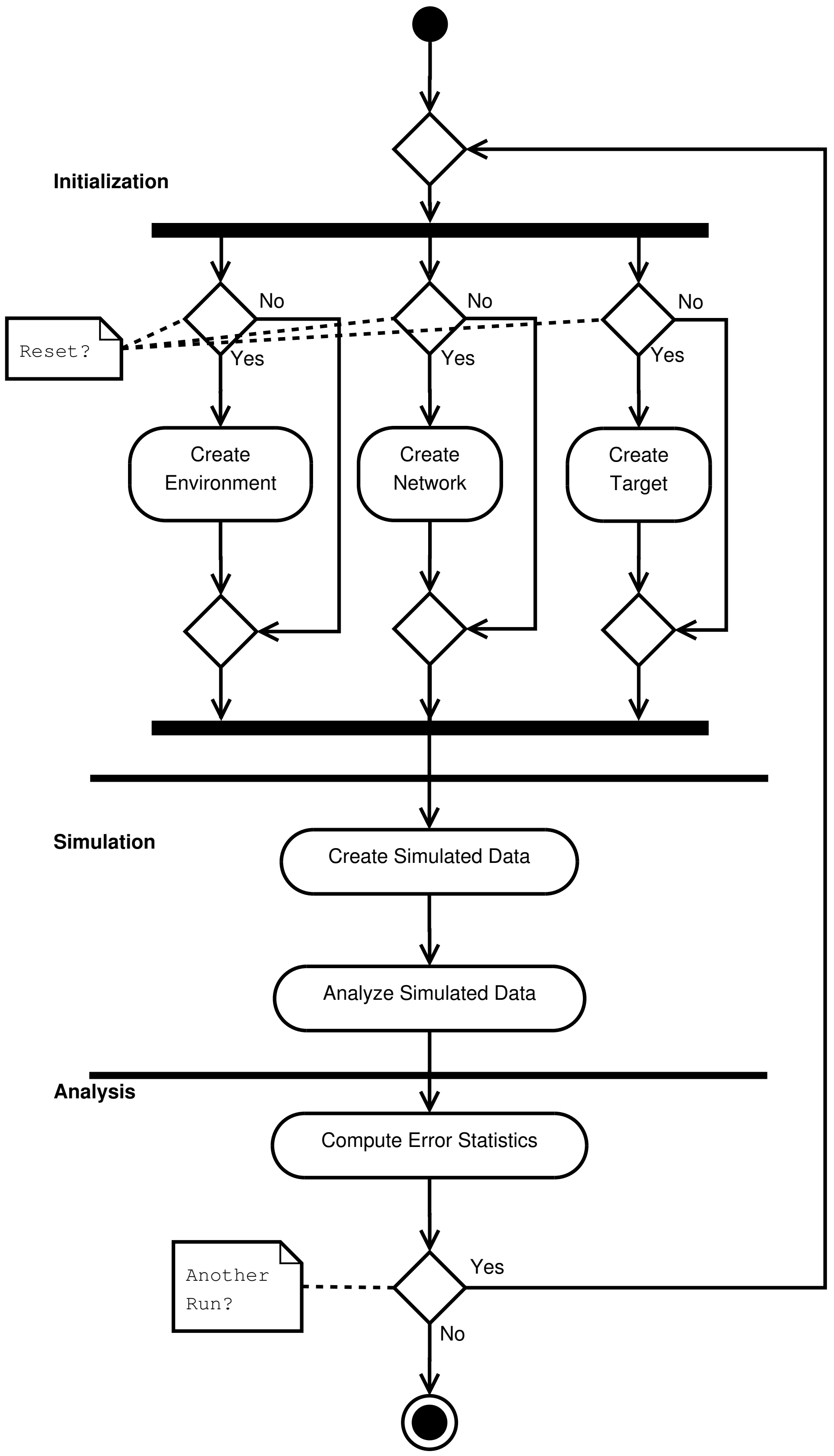

Figure 24. Activity diagram for a simulation scenario with multiple runs.

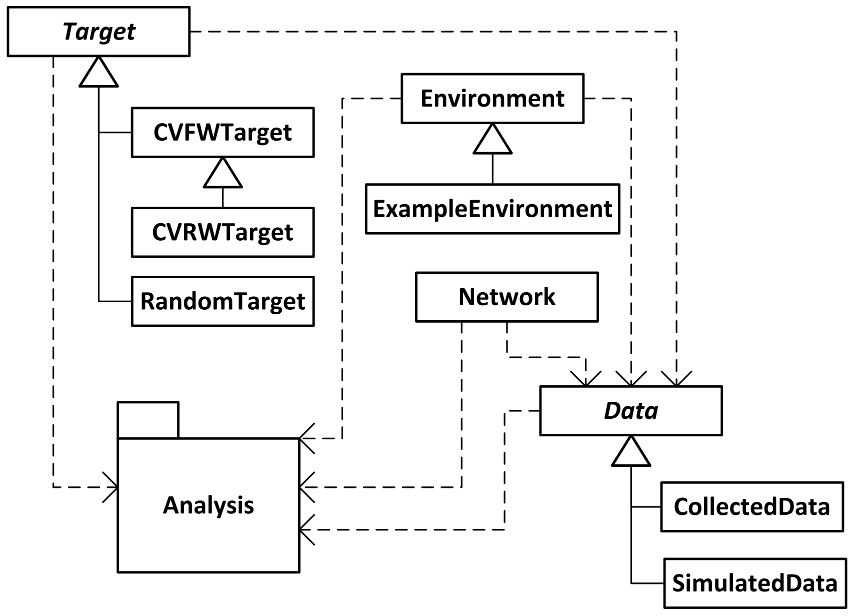

Figure 25. Structure diagram for the **Geolocation** package.

1. Environment Classes

As depicted in Figure 25, there are two classes that describe the local propagation environment. These are the **Environment** and **ExampleEnvironment** classes. The former provides a description of a uniform environment completely characterized by its total atmospheric pressure, temperature, dew-point, and path loss exponent. It also provides functionality to compute the index of refraction, refractivity, and tools to convert between the two values. The **ExampleEnvironment** class provides a small selection of predefined environments which may be called by name.

2. Network Class

The **Network** class encapsulates the parameters necessary to describe an IEEE 802.16 OFDMA compliant network [1]. These parameters are described in Chapter II. The constructor takes as arguments the primitive network parameters, and the derived parameters are computed upon demand by the appropriate associated "get" method.

3. Target Classes

The ***Target*** class encapsulates the necessary information about target state over time to facilitate creation of the simulated data sets. Three subclasses that enable efficient creation of targets with different basic profiles are provided. The **RandomTarget** class generates a specified number of randomly placed targets with random velocity vectors. The **CVFWTarget** class generates state information for a single target traversing a specified track at constant speed. This class does not enforce the maximum 35 s between ranging events required by the standard nor any other ranging triggers. The **CVRWTarget** class generates state information for a single target traversing a random set of way-point at constant speed.

4. Data Classes

The ***Data*** class and its associated subclasses provide three principle functionalities. First is the establishment of a common core of properties that may be taken as available by the classes associated with the **Analysis** subpackage. Second, it allows users to generate simulated data sets consistent with the information contained in specified input objects of classes **Environment**, **Network**, and ***Target***. Finally, it provides an interface structure allowing users to input data collected in field experiments in a way compatible with the requirements of the appropriate member of the **Analysis** subpackage. It should be noted that the option to mask data given in the **SimulatedData** class simulates intermittently available data by using a simple process that resets at random all but a subset of the data at any observation to NA. This is done without reference to how close or far the target is from any tower.

5. Analysis Subpackage

The **Analysis** subpackage contains the set of classes and supporting functions that enable various kinds of analysis of the kinds of data contained and described in objects of class ***Data***. A class diagram for the **Analysis** subpackage is given in Figure 26. It contains several component classes. The details of which classes in the **Analysis** subpackage depend

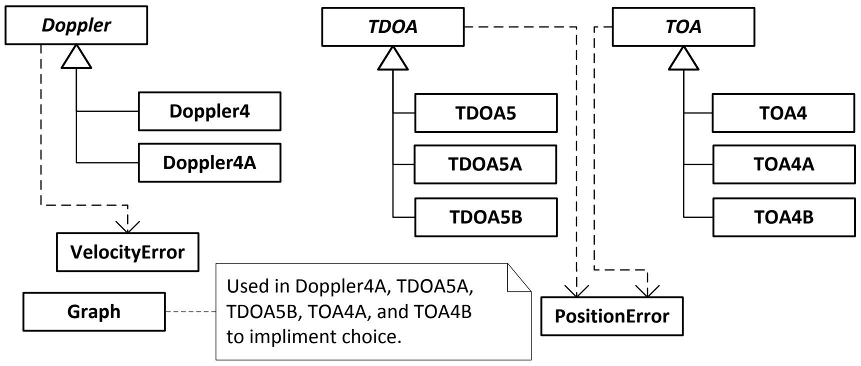

Figure 26. Structure diagram for the **Analysis** subpackage.

on which external classes in the larger **Geolocation** package are shown in the class structure diagram presented in Figure 27.

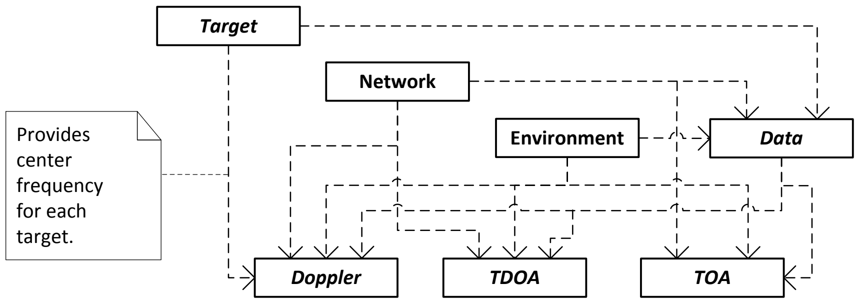

Figure 27. Structure diagram for the **Analysis** subpackage with external dependencies.

a. TDOA Class Family

Four classes in the TDOA family are shown in Figure 25. The base class, ***TDOA***, provides a common set of properties, an abstract method for constructing the constraint graph, and a concrete method for constructing individual constraint equations. The **TDOA5**, **TDOA5A**, and **TDOA5B** classes are derived from this common base. The **TDOA5** class implements Bakhoum's algorithm as described in [13] modified to reflect the assumption of a single, common value for refractivity in the environment and the need to handle the case of a negligible time difference of arrival between two observing receivers. The **TDOA5A** class implements a modified version of the algorithm used in the **TDOA5** class by incorporating receiver choice. The **TDOA5B** extends the **TDOA5A** class by incorporating the weighting function defined in Chapter IV.

The first implementation detail of note is that, for efficiency and simplicity, the **TDOA5A** and **TDOA5B** classes determine the two most orthogonal constraint vectors by finding the pair with the minimum absolute inner (dot) product rather than the cross product with the largest magnitude. This is because the latter first requires computing the cross product (a vector) followed by computing the length of the cross product vector, while the former eliminates the first step. All three concrete classes of this family handle the case of negligible time difference between two observing receivers by exploiting the edge weight setting functionality of the **Graph** class to give such edges zero weight. If this happens, the **TDOA5** class does not enforce does not enforce the requirement that the TDOA constraint graph takes the shape of a star (all hyperbolic constraints share a common receiver) as is assumed in [13]. In all cases **TDOA5** simply uses the first five edges of the tree generated by the `findTree` method of the **Graph** class to form the estimate.

b. TOA Class Family

Four classes in the TOA family are shown in Figure 25. The abstract base class, ***TOA***, provides a common set of properties, an abstract method for constructing the constraint graph, and a concrete method for constructing individual constraint equations. The **TOA4**, **TOA4A**, and **TOA4B** classes are derived from this common base. The **TOA4** class implements a time-of-arrival based algorithm employing four observing receivers to estimate the position of an emitter in three dimensions. This is a circular multilateration scheme. The **TOA4A** class extends the **TOA4** class by implementing constraint choice though use of the functionality of the **Graph** class. The **TOA4B** class extends the **TOA4A** class by implementing a method for weighting the edges of the TOA constraint graph.

The first implementation detail of note is that for efficiency and simplicity the **TOA4A** and **TOA4B** classes determine the two most orthogonal constraint vectors by finding the pair with the minimum absolute inner (dot) product rather than the cross product with the largest magnitude. This is because the latter first requires computing the cross product (a vector) followed by computing the length of the cross product vector whilst the former eliminates the first step. Second, this class does not check for the presence of either of the degeneracy conditions discussed in Chapter IV. This should be noted by anyone who decides to use these algorithms with a network where this condition is known to exist.

c. Doppler Class Family

Three classes in the Doppler family are shown in Figure 25. The base class, ***Doppler***, provides a common set of properties, an abstract method for constructing the constraint graph, and a concrete method for constructing individual constraint equations. The **Doppler4** class uses a Doppler-based approach to estimate the velocity of a mobile device in three dimensions using four observing receivers. The class assumes that the center frequency of transmission is known. The **Doppler4A** class implements a modified version of the algorithm used in the **Doppler4** by incorporating constraint choice though use of the functionality of the **Graph** class.

Two implementation details are of particular note. First, none of these classes check the condition that three receivers are co-linear which would result in degeneracy. This choice was made because it is reasonable to expect this to not be the case. Second, the **Doppler4A** class definition is designed such that it would be very easy to design a **Doppler4B** class should an appropriate weighting function be available. This is because the **Doppler4A** class includes a loop in the `constraintGraph` function that sets the weight of each edge to one. Finally, the Doppler family of classes depends on the ***Target*** class because the center frequency of transmission associated with any target is not included in the ***Data*** class. This reflects the assumption made in Chapter II that the said center frequency is known *a priori*.

d. PositionError Class

The goal of the **PositionError** class is to automate the generation and plotting of certain error statistics related to position estimates.

e. VelocityError Class

The goal of the **VelocityError** class is to automate the generation and plotting of certain error statistics related to velocity estimates.

f. Graph Class

The **Graph** class provides encapsulation for the information necessary to describe a graph and to find a maximum weight spanning tree on said graph. The choice of a maximum weight spanning tree was made in order to take advantage of weight as a measure of edge preferability. That is, edges with higher weights are preferred to edges with lower weights. It should be noted that the algorithm may be converted to a minimum weight spanning tree through the use of the inverse map on the set of edge weights.

A few implementation details are of note. First, while it is possible to directly supply a vector of edge weights, this is not the preferred approach. The `setEdge` and `getEdge` functions provide a reliable interface for setting individual edge weights and retrieving the weight associated with any edge. At this time, these functions are designed to only set or get a single weight; however, it would be possible to build on them to enable setting or getting weights in batches. Second, by convention, an edge weight of zero is considered to indicate "no edge" and edge weights must be positive. Third, the `findTree` method does not allow enforcement of a "star" topology for the output tree.

C. SIMULATION DESIGN

Monte Carlo simulations were designed to independently test the impact of receiver choice (Section 3) and refractivity (Section 4) upon the quality of position estimates. Several scenarios employing known targets tracks in space were also designed in order to take a first look at the impact of these variables upon the problem of tracking a moving target through a network (Section 5).

1. Exogenous Variables

The principle variables subject to choice are those required to construct the **Environment**, **Network**, and ***Target*** class objects employed in the analysis as well as the assumed refractivity of the medium provided to the position and/or velocity estimators. In order to make better comparisons, seeds for the random number generator were matched where appropriate.

2. Error Structure

The principle assumption made in formulating an error structure in this work is that the simulated ranging process will yield the best possible final, stable values. For time adjust, this means with resolution of one TA unit. For frequency adjust this means one Hz. This assumption is facilitated by assuming that the simulated network uses a GPS timing reference source per [1, 8.4.10.1]. GPS disciplined crystal oscillators are capable of less than one part in 10^{12} of Allan deviation [47, 48]. This implies the ability to generate signals at the base station at 10 GHz with accuracy of 0.01 Hz, or put another way, a clock with timing resolution of better than 10^{-12} seconds. This is more than sufficient to make the resolution of the timing and frequency units the dominant source of error. Errors are assumed to be unbiased except for possible refractivity effects.

3. Impact of Receiver Choice on Randomly Distributed Targets

In order to assess the power of receiver choice in application to hyperbolic multilateration, a single environmental condition with ambient temperature of 30 Celsius, dewpoint of 20 Celsius, and total pressure of 1000 millibar was chosen. The parameters common to all the algorithms are presented in Table 4. In all cases, the base algorithm uses the required number of receivers. For TOA4 and Doppler4, this is four receivers, while for TDOA5 this is five. For all others, consideration of at least two additional receivers is enforced with additional receivers used if the detection threshold allows. The bandwidth was chosen to allow comparison to further simulations undertaken with a different assumed speed of light.

Table 4. Common simulation parameters for assessing receiver choice.

Parameter	Value
Temperature (C)	30
Pressure (mb)	1000
Dew-point (C)	20
N (ppm)	True Value
Path Loss Exponent	2
Bandwidth (MHz)	100
Subcarriers	2048
Runs	500
Targets/Run	250
Receivers	50
Detection Threshold (dBm)	-90
X-Coordinate Limits (m)	$\pm 10\,000$
Y-Coordinate Limits (m)	$\pm 10\,000$
Z-Coordinate Limits (m)	± 100
Separation (m)	1000

4. Impact of Refractivity

In order to assess the impact of refractivity, an additional set of three scenarios were designed. These use the same common parameters as in the case of assessing receiver choice as given in Table 4. Unlike the previous case, the propagation velocity is assumed to be 3×10^8 m per second.

5. A First Look at the Tracking Problem

While the **Geolocation** package does not explicitly implement any tracking functionality, it is possible using the **CVFWTarget** class to take a first look at how some of the methods proposed in this thesis perform for a single target on a defined track. The test track and positions of the receivers in the simulated network are shown in Figure 28. Fifty sets of data were generated along the track shown at positions marked by circles. Each of the three TOA algorithms was applied to each of these data sets in turn and errors computed. The results are presented in Section D.3.

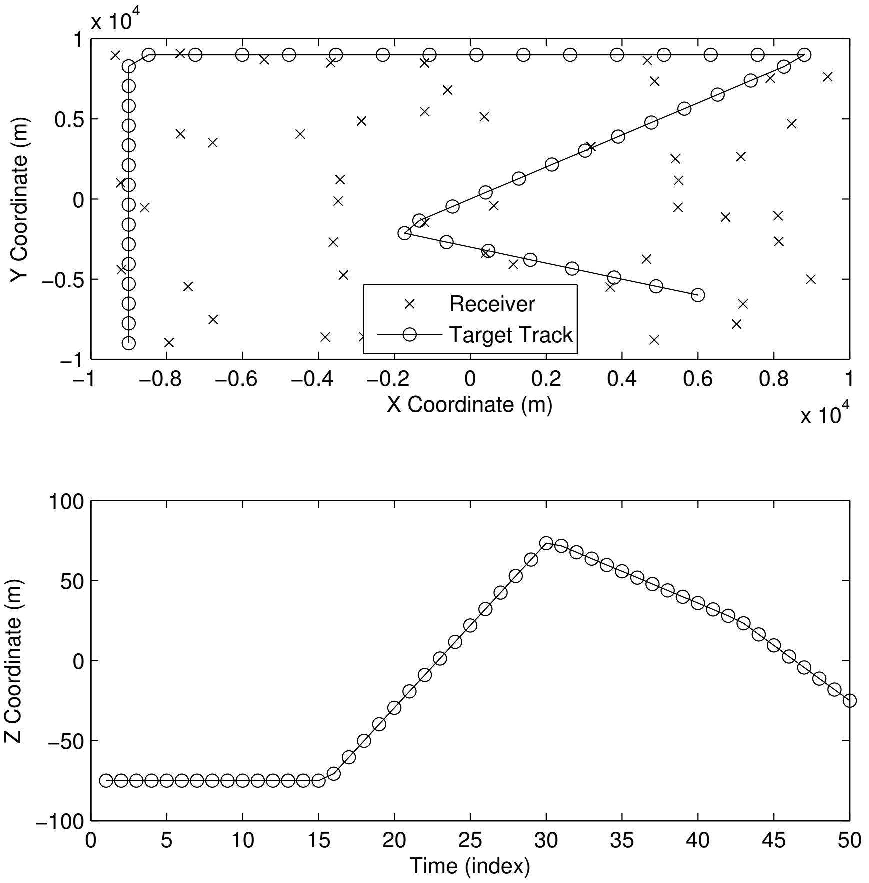

Figure 28. Test case for measuring the performance of TOA algorithms when applied to the tracking problem.

D. RESULTS

A presentation of the results of the simulation study undertaken is contained in this section. The results related to receiver choice, refractive effects, and the initial tracking scenario are presented in Subsections 1–3.

1. Receiver Choice

The impact of employing receiver choice varies across the different algorithms. The algorithms are summarized in Table 5. A qualitative summary of the impact across the different algorithms and measures based on observed patterns in the graphs of the kernel density estimates is provided in Table 6.

Table 5. Summary of the major features of the various algorithms.

Algorithm	Description	Comments
Doppler4	3D Doppler velocity estimator	Reference method.
Doppler4A	Doppler4 with unweighted receiver choice	All edges of constraint graph have weight one.
TOA4	3D time-of-arrival position estimator	Reference method. Extends [14]
TOA4A	TOA4 with unweighted receiver choice	All edges of constraint graph have weight one.
TOA4B	TOA4 with weighted receiver choice	Constraint graph edges weighted to prefer constraints close to one receiver or the other relative to the distance between receivers.
TDOA5	3D time-difference-of-arrival position estimator	Reference method. Extends [13]
TDOA5A	TDOA5 with unweighted receiver choice	All edges of constraint graph have weight one.
TDOA5B	TDOA5 with weighted receiver choice	Constraint graph edges weighted to prefer large time differences relative to distance between receivers.

Table 6. Summary of the impact of receiver choice on selected performance measures relative to the relevant baseline algorithm.

Algorithm	Mean	Median	SD	IQR
Doppler4A	Tighter confidence interval	Higher median, similar dispersion	Significant improvement	Higher median, tighter confidence interval
TOA4A	Narrow confidence interval concentrated at lower values	Narrow confidence interval concentrated at lower values	Narrow confidence interval concentrated at lower values	Narrow confidence interval concentrated at lower values
TOA4B	Narrowest confidence interval concentrated at lowest value.	Narrowest confidence interval concentrated at lowest value.	Narrowest confidence interval concentrated at lowest value.	Narrowest confidence interval concentrated at lowest value.
TDOA5A	Significant improvement	Significant improvement	Significant improvement	Significant improvement
TDOA5B	Similar to TDOA5A	Similar to TDOA5A	Similar to TDOA5A	Similar to TDOA5A

The Doppler4A algorithm is an example of where sensor choice has an inconsistent impact across measures. If mean and standard deviation are the primary performance metrics, then employment of choice is supported by the results. The plot of the density estimate for the mean L_2 error is shown in Figure 29. The median of both density functions is very similar and the number of cases in which the mean error is greater than 200 m is substantially reduced. This is in comparison to the estimated density function for the inter-quartile range of the L_2 errors, a plot of which is shown in Figure 30. While the dispersion of the inter-quartile range is visibly reduced, the median is higher.

The impact of receiver choice is consistent for the TDOA family of algorithms. Incorporating receiver choice significantly improves performance. The weight function proposed in Chapter IV does not appear to improve performance except for a small improvement in the median L_2 error. A plot of the kernel density estimate of the mean L_2 error is shown in Figure 31, and the standard deviation of the L_2 errors is shown in Figure 32. In both cases, the density associated with the TDOA5A and TDOA5B algorithms is

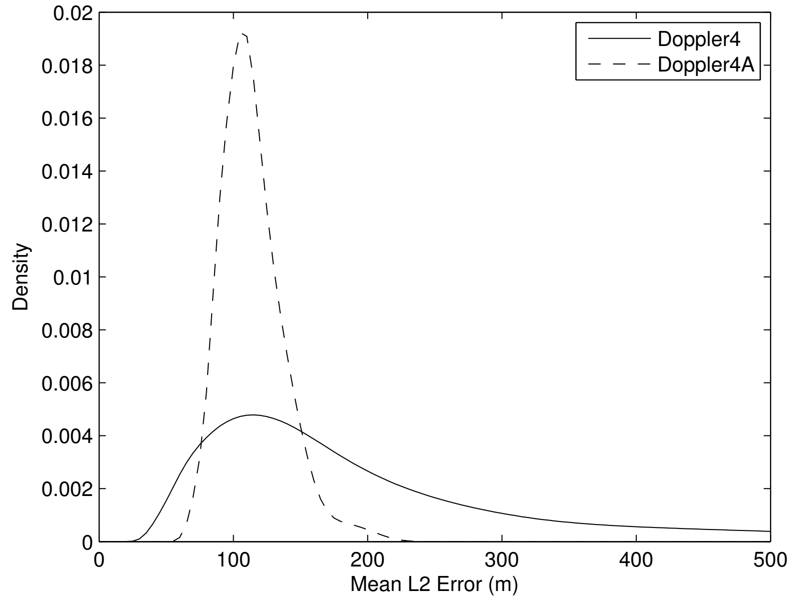

Figure 29. Kernel density estimate of the mean L_2 error when using the Doppler4 and Doppler4A algorithms.

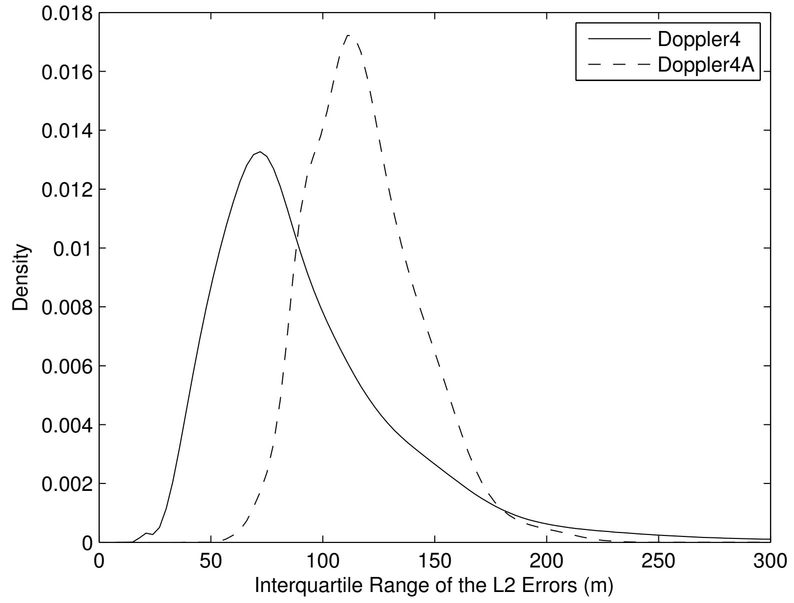

Figure 30. Kernel density estimate of the inter-quartile range of L_2 errors when using the Doppler4 and Doppler4A algorithms.

concentrated at lower values, while the TDOA5 algorithm has significant amounts of density at higher values. However, there is not a significant difference between the TDOA5A and TDOA5B algorithms. In addition, while the TDOA5B algorithm may slightly outperform the TDOA5A algorithm in terms of the mean L_2 error, the reverse appears to be true for the standard deviation of the L_2 errors.

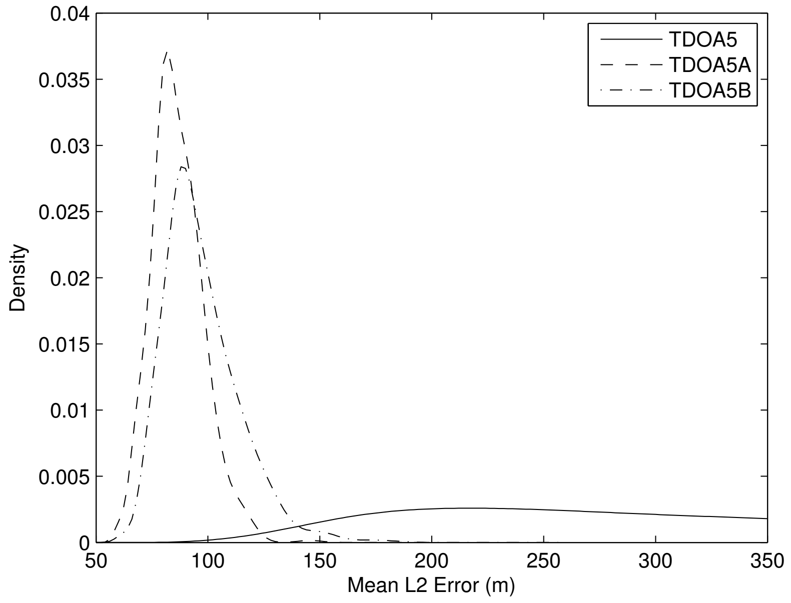

Figure 31. Kernel density estimate of the mean L_2 error when using the TDOA5, TDOA5A, and TDOA5B algorithms.

The results for the TOA family of algorithms may be summarized as choice helps and the suggested weight function used in the TOA4B method provides further performance gains on average. A plot of the kernel density estimates for the median and interquartile range of the L_2 errors are shown in Figures 33 and 34, respectively. These two plots show the statistics for which the improvement was least. In both cases, addition of unweighted choice to the algorithm provides a significant improvement in performance as indicated by the much taller peak centered at a lower value compared to the baseline. The addition of weights that favor constraint planes closest to one receiver further narrows the confidence intervals and reduces the median value of each statistic.

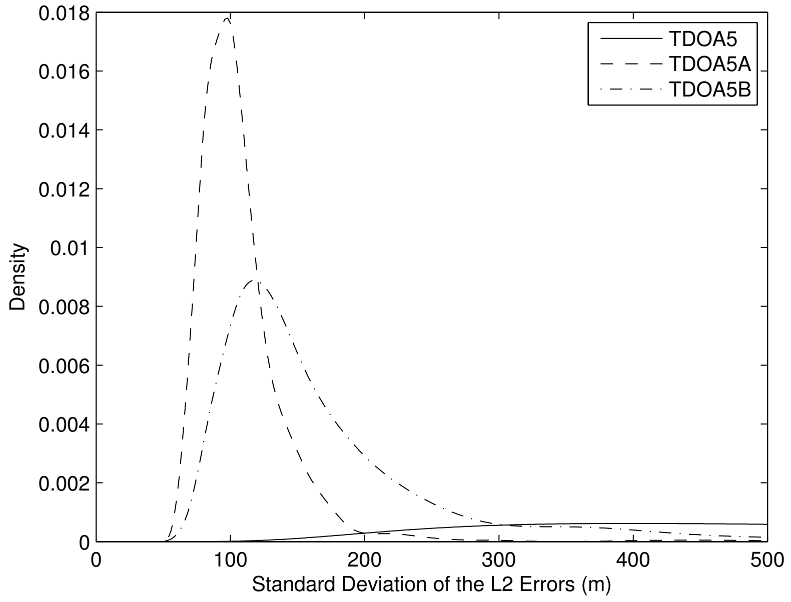

Figure 32. Kernel density estimate of the standard deviation of L_2 errors when using the TDOA5, TDOA5A, and TDOA5B algorithms.

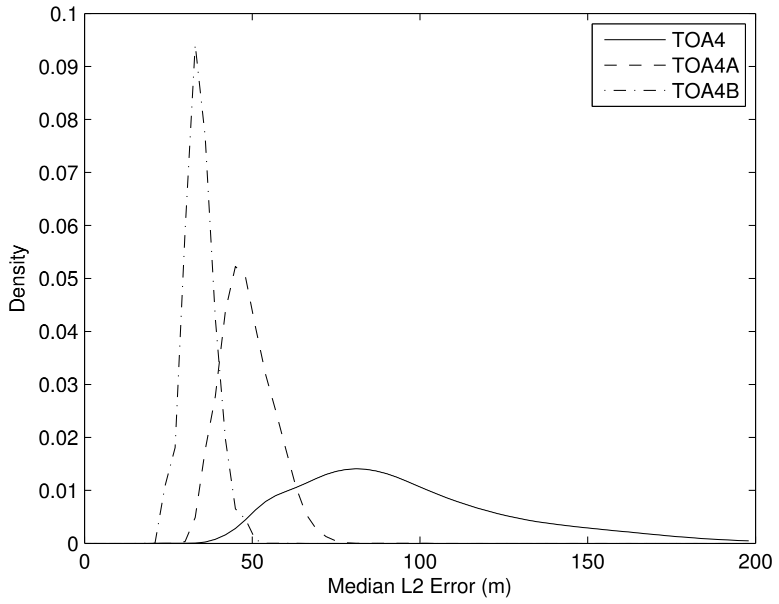

Figure 33. Kernel density estimate of the median L_2 error when using the TOA4, TOA4A, and TOA4B algorithms.

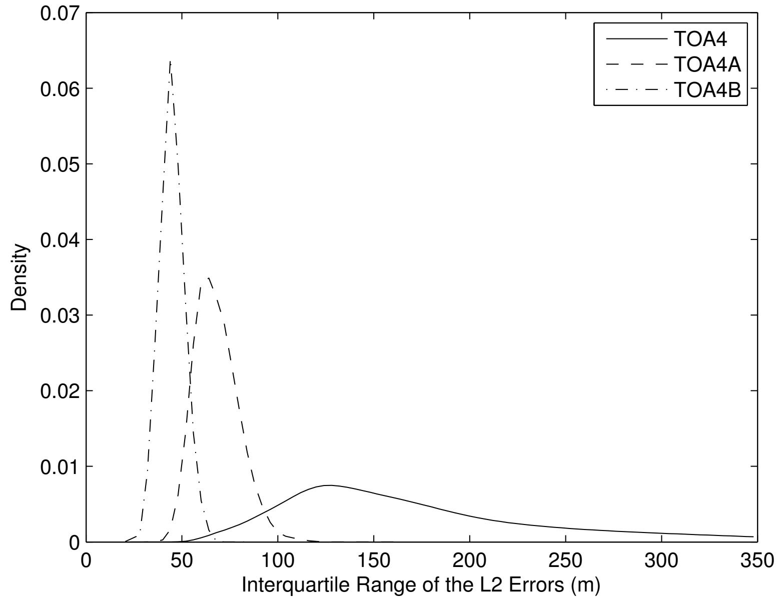

Figure 34. Kernel density estimate of the inter-quartile range of L_2 errors when using the TOA4, TOA4A, and TOA4B algorithms.

2. Refractivity Effects

When estimating velocity from FA information, there is no apparent benefit from accounting for refractive effects. The estimated density function for the mean L_2 error, a plot of which is shown in Figure 35, is a representative example. The curves for each estimated density function are virtually indistinguishable.

When using TDOA methods to estimate position, choice of refractivity has a complicated impact. A plot of the kernel density estimates of for the mean L_2 error when employing the TDOA5B algorithm with two different assumed values of refractivity is shown in Figure 36. In this case, assuming a refractivity of $N = -692$ significantly degrades performance.

The results of considering refractive effects in the context of TOA algorithms are consistent. In all cases, assuming the speed of light is 3×10^8 m/s noticeably (and in some cases significantly) decreases performance. One good example is the mean L_2 error; a plot of the density estimate for which is shown in Figure 37. Employing either the TOA4A or

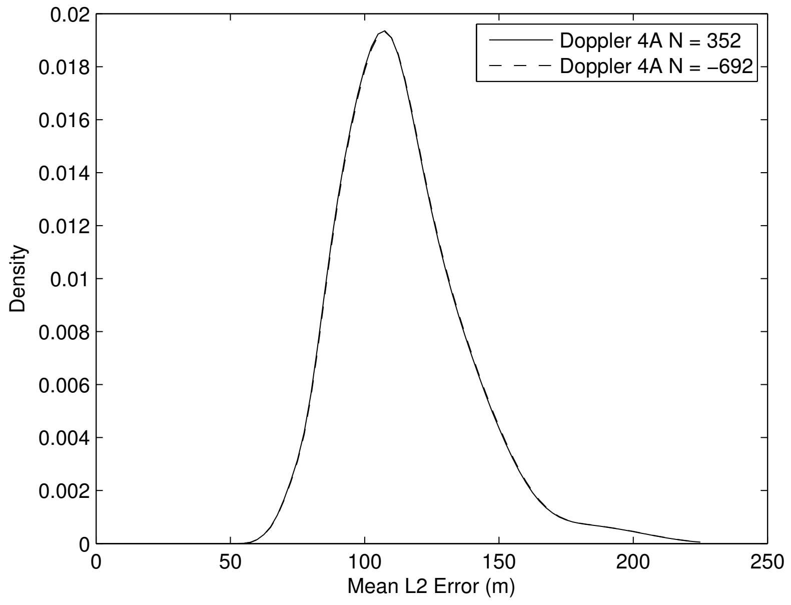

Figure 35. Kernel density estimate of the mean L_2 error at two different refractivity values when estimating velocity in a 100 MHz network.

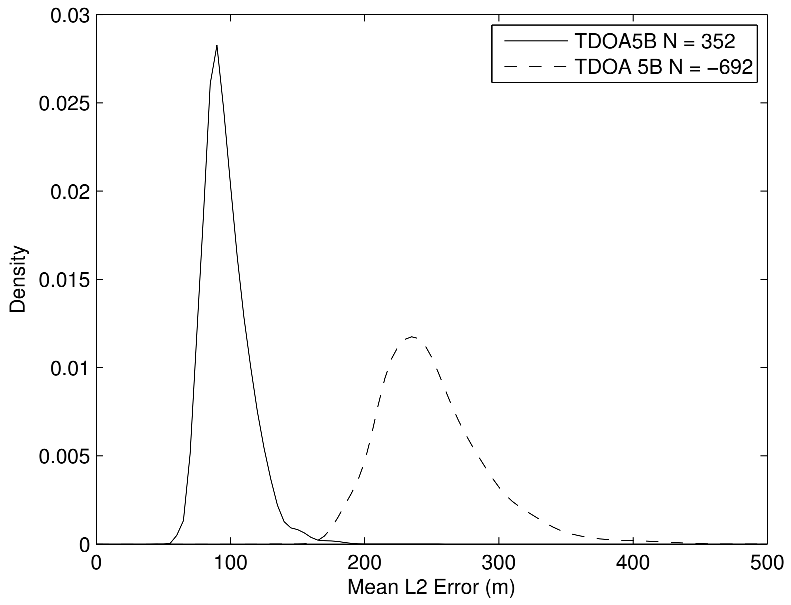

Figure 36. Kernel density estimate of the mean L_2 error at two different refractivity values when estimating position using the TDOA5B algorithms in a 100 MHz network.

TOA4B algorithm, we see that using the correct propagation velocity estimate improves performance. The plot of the density estimates for the standard deviation of the L_2 errors shown in Figure 38 shows a similar pattern. In this figure, the TOA4B algorithm does not appear to perform much worse when a speed of light of 3×10^8 m/s is specified than the TOA4A algorithm does with the correct refractivity.

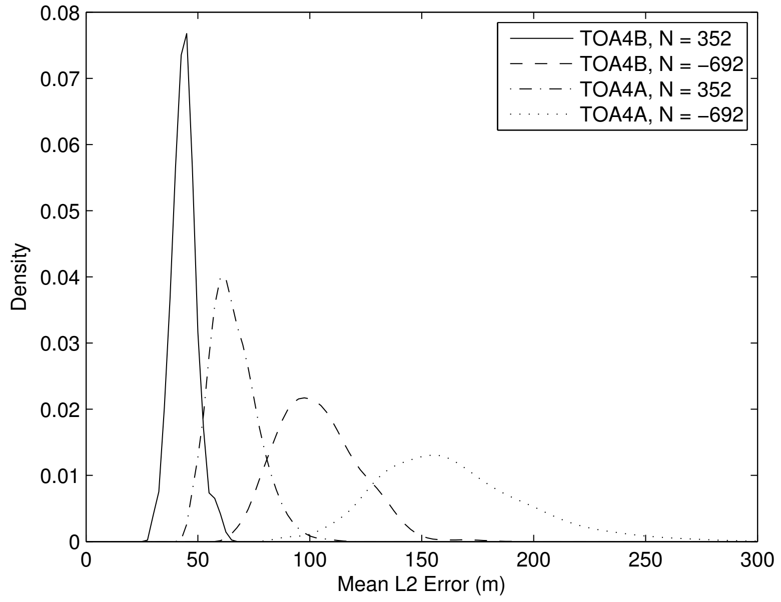

Figure 37. Kernel density estimate of the mean L_2 error at two different refractivity values when estimating position using the TOA4A and TOA4B algorithms in a 100 MHz network.

3. Tracking

The results for the tracking test case when using the TOA family of algorithms is not completely comparable to the previous results because for these the choice was made to compute the L_2 errors in the x-y plane and ignore the z-dimension error. Plotting a kernel density estimate of the two-dimensional errors showed clear improvement in the precision of the estimates with the TOA4B algorithm outperforming the TOA4A algorithm which in turn outperforms the TOA4 algorithm as shown in Figure 39. As this is a tracking scenario, it is also of interest to look at the same errors with respect to time as shown in

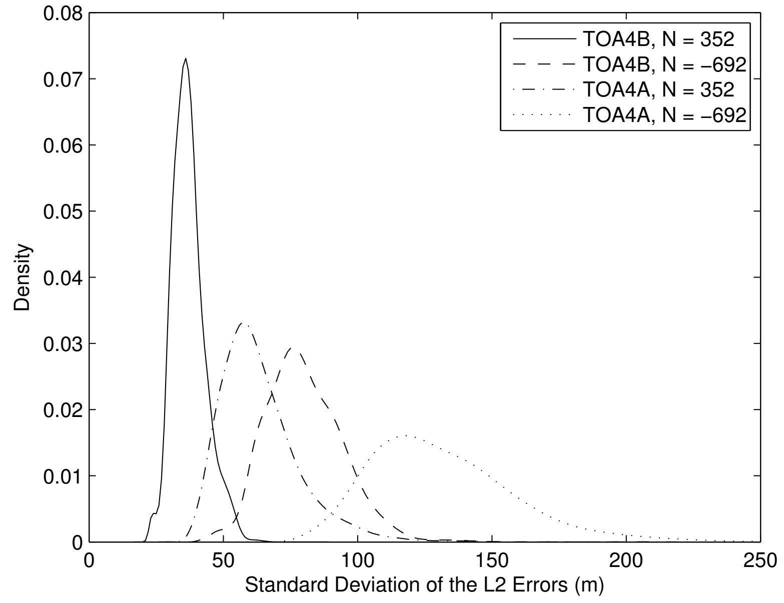

Figure 38. Kernel density estimate of the standard deviation of the L_2 error at two different refractivity values when estimating position using the TOA4A and TOA4B algorithms in a 100 MHz network.

Figure 40. There are periods during the tracking scenario when all three methods perform approximately equally well, especially very early and between times twelve and twenty-one. The TOA4B algorithm is almost always best, but the margin is usually small relative to the TOA4A algorithm. The TOA4 algorithm exhibits several large error spikes. Given the way in which receivers are allocated, the bursty pattern may be because data from nearby track positions is likely being analyzed by the same four receivers; thus, if the receiver geometry is poor, it will be poor for a group of nearby points. As the other algorithms force inclusion of two additional receivers, this likely accounts for the better performance of the other algorithms as the choice algorithm will in general try to choose a better set of receivers. During the spikes in TOA4 position errors, the TOA4A and TOA4B position errors remain low and are consistent with the TOA4A and TOA4B position errors elsewhere (for instance, position estimates one through eight and 20 through 30, inclusive).

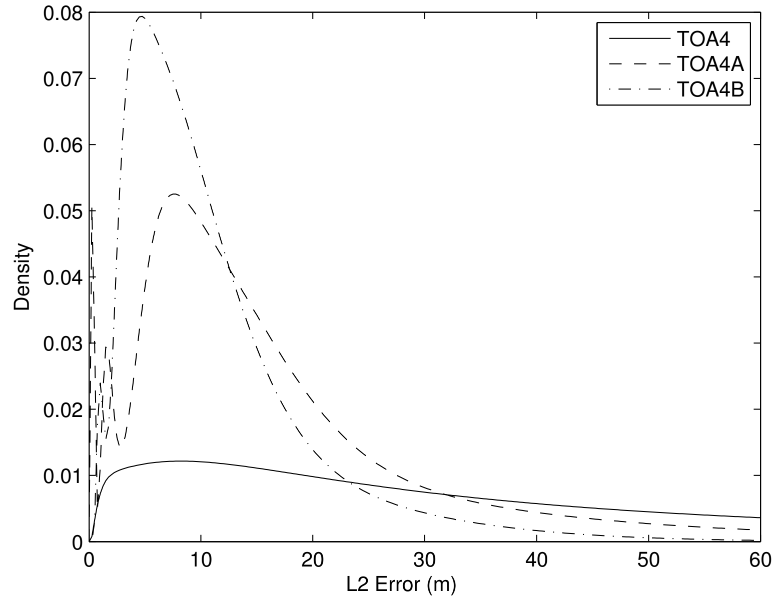

Figure 39. Kernel density estimate of the L_2 errors in the x-y plane when using the TOA4, TOA4A, and TOA4B algorithms to track a target.

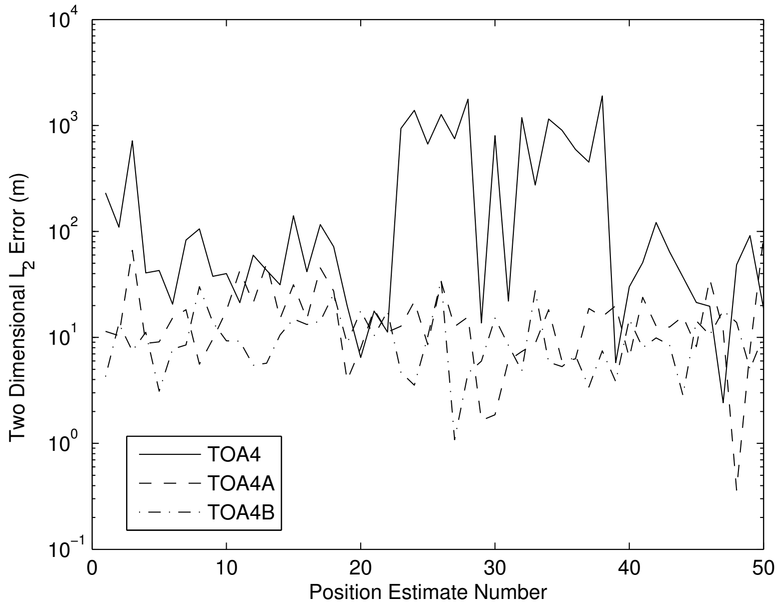

Figure 40. The L_2 error in the x-y plane when using the TOA4, TOA4A, and TOA4B algorithms to track a single target as a function of time.

E. DISCUSSION

The various position and velocity solver classes do not check for relative receiver positions which would cause degeneracy problems other than the check in the TDOA algorithms for a negligible time difference-of-arrival. When using randomly distributed receiver positions in three space, the probability of such an occurrence is negligible given the use of one-m resolution in each dimension. A user could conceivably manually enter a network configuration with receiver positions where this problem arises.

As noted in Section B.5.a, the **TDOA5** class does not strictly follow the algorithm given in [13] when the time difference-of-arrival between two receivers is zero. Due to the way in which the `findTree` method is implemented in the **Graph** class, this is only an issue if the tie is between the the lowest-numbered receiver and any of the others. Moreover, such ties tend to be rare, especially when the bandwidth of the simulated network is high. As the goal of the **TDOA5** class is to provide a naive baseline solution to which to compare the modified algorithms implemented in **TDOA5A** and **TDOA5B**, this small deviation does not impact the validity of the result.

It is important to note the impact of the computational complexities of the various algorithms. The approximate computational time of the various simulations in hours are reported in Table 7. Due to the use of common parameters, any given scenario can be identified by an algorithm/refractivity pair. As is expected, there is a significant time penalty paid for the use of choice in any form. The baseline algorithms were not significantly faster using **TOA4** and **Doppler4** because of the design choice to use the methods implemented in the **Graph** class to transform the raw set of receiver indices into a set of indices that could be used to compute the final solution. The same method was also used in the **TDOA5** class to protect against cases in which there is a negligible time difference-of-arrival between any pair of receivers, thus, avoiding numerical degeneracy in the solution.

The simulation results validate that choice is a powerful technique for improving the performance of the various estimators. In most cases, the estimator employing the naive choice algorithm outperforms the baseline estimator. It is less clear that incorpo-

Table 7. Approximate computation times for random target scenarios.

Algorithm	N	Time
TOA4	True	18h
TOA4A	True	18h
TOA4A	-692	18h
TOA4B	True	18h
TOA4B	-692	18h
TDOA5	True	18h
TDOA5A	True	7d
TDOA5B	True	7d
TDOA5B	-692	7d
Doppler4	True	18h
Doppler4A	True	18h
Doppler4A	-692	18h

rating additional *a priori* information about the constraints provides additional gains. For example, incorporating the proposed TDOA weighting scheme does not improve performance and may degrade performance by some measures. The TOA4B estimator provides a clear example of a good choice of weight improving performance.

The simulation results also validate that at sufficiently high system bandwidths (sufficiently high time degrees of time resolution), failure to incorporate refractivity can make a major impact on performance. Estimates of velocity do not seem to be impacted by the inclusion of refractive effects. In TOA and TDOA approaches, there is clear evidence that failure to choose a good refractivity estimate can have major negative impacts on estimator performance; although, the size of the impact seems to vary with the statistic of interest.

The one example of tracking using the three TOA algorithms, which considered two-dimensional rather than three-dimensional errors, is interesting for two reasons. First, it illustrates position errors using the TOA4 algorithm may exhibit bursty behavior. That is, there are distinct periods in which the TOA4 algorithm performs well and others in which it performs poorly. This behavior is not exhibited by the TOA4A or TOA4B algorithms, possibly because the inclusion of additional receivers in the constraint choice process allows these algorithms to compensate for the poor receiver geometries that are suspected to cause the problems exhibited by the TOA4 algorithm. Second, while the TOA4A algorithm outperforms the other TOA algorithms in three-dimensional estimation problems, in two dimensions the TOA4B algorithm may perform better.

The simulation framework and the results of the simulation study were presented in this chapter. The simulation framework takes the form of an object-oriented MATLAB package called **Geolocation** with its array of constituent classes and subclasses. Monte Carlo simulation study using this package provided proof-of-concept of both the utility (under appropriate circumstances) of incorporating refractive effects and receiver choice.

VI. CONCLUSION

The problem of how to refine methods for the passive geolocation of emitters in an IEEE 802.16 OFDMA compliant wireless network was addressed in this thesis. The two specific problems studied in this thesis were to incorporate the effects of local refractivity and to exploit the ability to choose from the available receivers those that are used to form the final position or velocity estimate. The former is applicable to all algorithms that require an estimate of the propagation velocity of the received signal. The latter is applicable to any estimator whose final form is a matrix equation and employs exactly three constraint equations.

A. SUMMARY OF WORK

Ways in which to refine three common methods for extracting position or velocity estimates from timing information in an IEEE 802.16 OFDMA network were explored in this thesis. The first is the incorporation of clear-air refractive effects. The second is the employment of receiver (and correspondingly constraint) choice to improve the performance of the base estimators.

Clear air refractive effects were incorporated into the positioning methods by examining their impact on the local propagation velocity. The local propagation velocity is a function of refractive index which may be computed from four atmospheric variables: ambient temperature, total pressure, dew-point, and carbon dioxide content.

Choice of constraints was incorporated though the use of a graph theory model. In this model, the vertices correspond to the towers that have relevant information about the target emitter and the edges represent possible constraints (circular, hyperbolic, or linear). In the case of TOA and TDOA, this graph is used to identify a set of independent constraints from which the final solution may be constructed. In the case of DVE, the set of constraints is not independent but is the smallest subset which incorporates information from all the available receivers. An attempt was made to use weighted edges, but the proposed weight functions do not improve performance.

A MATLAB package called **Geolocation** was developed to provide end-to-end simulation capability. It includes classes to describe the environment, network configuration, and target behavior. It provides other classes to generate data, estimate either position or velocity in three dimensions, and generate the error vectors.

Simulation study was undertaken to test the efficacy of the aforementioned modifications. Incorporation of receiver choice has a dramatic impact on the performance of the geolocation algorithms. Refractivity has a much less significant effect.

B. SIGNIFICANT RESULTS

Several significant results were presented in this thesis. These are the incorporation of refractive effects, the development of an algorithm for receiver choice, and an object-oriented MATLAB package for conducting simulations.

A method for incorporating refractive effects into geolocation models was proposed in this thesis. A decision aid for analysts in deciding whether or not to incorporate refractive effects, which relates the amount of bias per 10 μs to the standard deviation of the timing-based range errors as a function of refractivity and time resolution, is given in Figure 17.

A generic receiver choice algorithm and applications to three different geolocation algorithms were proposed. The proposed method uses a simple, linear algebra-based decision rule to choose constraint equations that together may be expected to be better conditioned than the naive choice made in the original algorithms taken from the literature. This work is believed to be new to the geolocation literature. Simulation results show unweighted receiver choice can significantly improve the performance of DVE, TOA, and TDOA schemes. Addition of an unweighted receiver choice to the TOA and TDOA algorithms yields 76% improvement of the median mean error for both. A 34% improvement in the median mean error is obtained by the addition of an unweighted receiver choice to the DD algorithm. The median standard deviations of the errors are improved by 91%, 91%, and 75%, respectively. The cost in terms of computational complexity for the use of this method varies by algorithm. For TOA and DD, the cost is on the order of the square of the number of receivers. For TDOA, it is on the order of the number of receivers to the fourth power.

An extensible, object-oriented MATLAB package was developed to conduct the simulation studies. It has component classes that describe the environment, the network configuration, the target (including three mobility models), build the simulated data, and conduct the various kinds of analysis. Due to the way in which it was designed, it is possible to build on the existing class definitions to create new algorithms. For instance, because the generic constraint equations are given in vector form, it is possible to build on the base classes for each algorithm family to construct two-dimensional estimators.

C. FUTURE WORK

There are many possible lines of future work either to increase the level of realism of the model or extend what is know or can be done with the available data.

The proposed family of classes that describe target behavior included methods for generating randomly positioned targets, a single target that moves along a fixed track at constant speed, and a single target that moves along a random track at constant speed. The **RandomTarget** class uses a naive mobility model to generate random velocity vectors by merely specifying an upper bound on the absolute value of any component. This could be improved upon by replacing it with a more realistic mobility model. The **CVFWTarget** class generates N data points evenly spaced in time. This could be improved upon by incorporating a mechanism to enforce network-specific ranging event triggers. While this class does require the user to input way points (and thus may model any terrestrial road network) it would be of use to build a class which inherits from **CVFWTarget** and provide the user a set of pre-defined road network models. There is no target model that allows non-constant speed targets. This is an area for improvement, especially if a class is developed to allow for different legs with different speeds, such as might be encountered on a real road network. The ***SimulatedData*** class employs a crude data masking scheme. Future work could explore how to perform more realistic data masking.

Position and velocity estimators to provide single-time-point estimates given a sufficiency of receivers are proposed in this thesis. As for the former, it would be of use to build classes to provide tracking functionality. This requires implementing a mechanism for storing current and historical target state information, a mechanism to estimate time-advanced timing or frequency information per the rules of dead reckoning, and a method for smoothing various position and velocity estimates. The second of these could be done by exploiting a ***Data*** class that provides an interface for formatting user data for analysis by other classes.

Only three-dimensional estimators with the associated cost of an additional required receiver over their respective two-dimensional counterparts were proposed. Two-dimensional estimators could be added to the **Analysis** subpackage using the tools provided in the ***TOA***, ***TDOA***, and ***Doppler*** classes.

Three sets of estimators expressed as matrix equations were proposed in this thesis. No estimate quality metrics were proposed. One possible avenue of research is to explore the eigenvalues of the $\mathbf{A}$ matrix of an estimator to assess estimate quality.

A method for incorporating clear-air refractive effects was proposed. The work is limited in that, while a possible metric by which to assess the limits of the point estimate is proposed, no thresholds have been identified. Future work could run a comprehensive set of simulation experiments to determine possible threshold values.

Weight functions for use in choosing constraints for building TOA and TDOA position estimates were proposed. The proposed TDOA weight function was demonstrated to be sub-optimal despite its initial intuitive appeal. Therefore, it would be helpful for future work to propose and test additional possible weight functions to obtain improved performance under most circumstances. Future work could also explore ways in which to extend the idea of a weighted constraint graph to the DVE case.

All of the proof-of-concept work in this thesis was performed via simulation. Simulation results come with the caveat that the assumptions of the simulation are reasonably reflective of real world conditions. Future work should apply the methods proposed in this thesis and implemented in software to data collected in the real world.

THIS PAGE INTENTIONALLY LEFT BLANK

A. A BRIEF INTRODUCTION TO UML DIAGRAMS

A brief introduction to reading Unified Modeling Language (UML) diagrams is provided in this appendix. Definitions, notations, and conventions are taken from [46] unless otherwise noted. Two kinds of diagrams are introduced in this appendix: Structure Diagrams and Activity Diagrams.

The note symbology is common to all kinds of UML diagrams. An example note is shown in Figure 41. The dashed line "tail" connects the note to whatever other symbol is

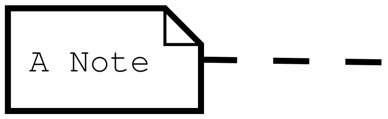

Figure 41. An example UML note with connector.

being commented on [46]. Here the note simply reads "A Note". The tail is optional.

A. CLASS DIAGRAMS

Class diagrams are used to model the relationships between different classes [46, p. 11]. An example of a class diagram is shown in Figure 42. This example shows three

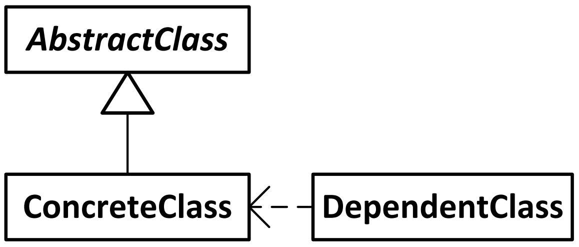

Figure 42. An example class diagram.

related classes, each shown as a box with a bold-face name in it. The arrows depict the two kinds of relationship used in this thesis. The kind connecting **ConcreteClass** to ***AbstractClass*** shows that the former is a generalization of the latter and inherits all properties and methods[46, p. 88]. The arrow connecting **DependentClass** to **ConcreteClass** shows that the former depends on the latter [46, p. 84]. Had the **DependentClass** class been dependent upon ***AbstractClass***, this would mean that it depends only on those parts of **ConcereteClass**, which are defined in ***AbstractClass***. If the name of a class is italicized, then it is an abstract class. This means that it has either properties and/or methods with defined names whose details of implementation are defined elsewhere [49, p. 10.78]. The **ConcereteClass** class provides a complete implementation of any abstract properties or methods described in ***AbstractClass*** as required to accomplish its particular purposes [46, p. 93].

B. ACTIVITY DIAGRAMS

Activity Diagrams are used to model sequential and parallel activities within the system [46, p. 11]. These diagrams always start with a filled circle and end with a filled circle within another circle [46, p. 44]. Diamonds denote decision and merge points [46, p. 47]. Fork and join bars (one-to-many and many-to-one) denote activities conducted in parallel [46, p. 50]. An activity diagram may use partitions (black lines) which divide the activity into groups of related processes [46, p. 59]. An annotated example of an activity diagram is shown in Figure 43. In this example, upon starting the activity there is an initial decision of whether to do Activity C or to flow into parallel execution of Activities A and B. After one of these two paths is taken, both merge into a single path which terminates at the output. An example partition is shown with an annotation. Normally this annotation would be the name of some region of interest in the activity.

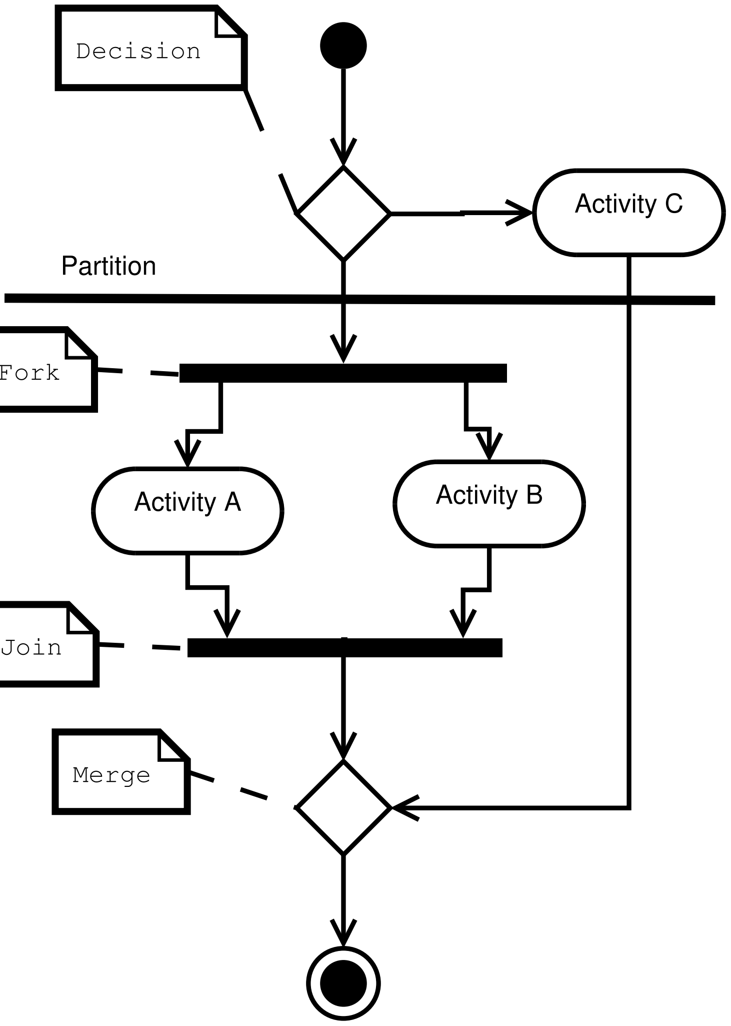

Figure 43. An example activity diagram.

THIS PAGE INTENTIONALLY LEFT BLANK

B. MATLAB CODE IMPLEMENTING THE *GEOLOCATION* PACKAGE

A. ENVIRONMENT CLASS FAMILY

1. Environment Class

```
classdef Environment < handle
   %ENVIRONMENT environmental model
   %   holds the data and provides methods for the physical
   %   model of the environment.
   %INPUTS to Environment
   %  pressure: total pressure in millibar
   %  temp: ambient temperature in degrees C
   %  dewpoint: dewpoint spread in degrees C
   %  pathLossExponent: the path loss exponent to be used
   %    in the one-way range equation.
   %CONSTANTS
   %  c: the NIST standard value for the speed of light
   %     299792458 m/s.
   %DEPENDANT Variables
   %  N: refractivity in parts per million.
   %  n: index of refraction (unitless)
   %STATIC Methods
   %  n2ppm(n): convert index of refraction n to
   %    refractivity
   %  ppm2n(N): convert refractivity N to index of
   %    refraction

   properties(Constant)
      c = 299792458;%speed of light in meters per second
   end%end constant properties

   properties (SetAccess = immutable)
      pressure = 1000%total atmospheric pressure in millibar
      temp = 25;%ambient temperature in degrees C
      dewpoint = 10;%dewpoint in degrees C
      pathLossExponent = 2;%path loss exponent
   end%end properties

   properties (Dependent)
   N%refractivity
   n%index of refraction
end

methods
   function[obj] = Environment(pressure, temp,...
         dewpoint, pathLossExponent)
      %validate and store pressure
      validateattributes(pressure, {'numeric'}, ...
         {'scalar', 'nonnegative'},'','pressure')
      obj.pressure = pressure;
      %validate and store temp
      validateattributes(temp, {'numeric'}, ...
         {'scalar','>=',-20,'<=',50},'','temperature')
      obj.temp = temp;
      %validate and store dewpoint
      validateattributes(dewpoint, {'numeric'}, ...
         {'scalar','<=',temp},'','dewpoint')
      obj.dewpoint = dewpoint;
      %validate and store pathLossExponent
      validateattributes(pathLossExponent, ...
         {'numeric'},{'scalar','positive'},'',...
         'pathLossExponent')
      obj.pathLossExponent = pathLossExponent;
   end

   function [out] =  get.N(obj)
      % convert the total pressure to dry air and water
      % vapor pressure
      Pw = parPresH2O(obj.pressure, obj.temp,...
         obj.dewpoint); %partial pressure of water
      Pd = obj.pressure - Pw;%dry air is what is left

      % Compute the refractivity estimate
      K1 = 77.6890;%dry air coeffient
      K2 = 71.2952;%wet air linear coefficient
      K3 = 375463;%wet air second term coefficient
      DK = obj.temp+273.15;%convert temperature to
                           %degrees kelvin
      out = K1*Pd/DK + K2*Pw/DK + K3*Pw/(DK^2);
   end
```

```
      function [out] =  get.n(obj)
         out = obj.ppm2n(obj.N);
      end

      %getRefractivity(obj)--returns refractivity of local
      %  environment
      %obj: environment for which to return
      %  refractivity
      function[N] = getRefractivity(obj)
         % convert the total pressure to dry air and water
         % vapor pressure
         Pw = parPresH2O(obj.pressure, obj.temp,...
            obj.dewpoint); %partial pressure of water
         Pd = obj.pressure - Pw;%dry air is what is left

         % Compute the refractivity estimate
         K1 = 77.6890;%dry air coeffient
         K2 = 71.2952;%wet air linear coefficient
         K3 = 375463;%wet air second term coefficient
         DK = obj.temp+273.15;%convert temperature
         % to degrees kelvin
         N = K1*Pd/DK + K2*Pw/DK + K3*Pw/(DK^2);
      end%end getRefractivity
   end%end methods

   methods (Static)
      function[N] = n2ppm(n)
         N = (n-1)*10^6;
      end

      function[n] = ppm2n(N)
         n = 1 + (N/(10^6));
      end
   end
end%end Environment class
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%parPresH2O.m -- Partial Pressure of Water in the atmosphere
%
%J. Q. McClintic, 2012
%
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius
%   dewpoint: current dewpoint temperature in celcius
%
%Outputs: PPW: partial pressure of water in millibars
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = parPresH2O(ATP, temp, dewpoint)
    % Temperature Check
    % If the temp is greater than 0 deg C, use the water
    % curve, else the ice curve
    if temp > 0
        PPW = waterCurve(ATP, dewpoint);
    else
        PPW = iceCurve(ATP, dewpoint);
    end %end if
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%waterCurve.m -- A function to compute the partial pressure
%  of water assuming the air temperature is greater than 0
%  degrees C
%
%Buck, A. New Equations for Computing Vapor Pressure and
%Enchancement Factor. Journal Of Applied Meteorology.
%December 1981, 1527-32.
%
%J. Q. McClintic, 2012
%
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius. ambient for pure water
%     vapor, else dewpoint temperature
%
%Outputs: PPW: partial pressure of water in millibars
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = waterCurve(ATP, temp)
    % Compute the unenchanced partial pressure of water

    %declare the various parameters table 2, curve ew6
    a = 6.1121;
    b = 18.564;
    c = 255.57;
    d = 254.4;

    % compute the partial pressure
    Ew = a*exp( temp*(b - temp/d)/(temp + c) );

    % Compute the enhancement factor

    % declare the various parameters
    A = 7.2e-4;
    B = 3.2e-6;
    C = 5.9e-10;
    D = 0; %included in case the choice of
    E = 0; % cuves were to change

    % compute the enhancement factor
    f = 1 + A + ATP*(B + C*(temp + D + E*ATP)^2);

    % Compute the full partial pressure
    PPW = Ew*f;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%iceCurve.m -- A function to compute the partial pressure of
%   water assuming the air temperature is less than 0
%   degrees C
%
%Buck, A. New Equations for Computing Vapor Pressure and
%Enchancement Factor. Journal Of Applied Meteorology.
%December 1981, 1527-32.
%
%J. Q. McClintic, 2012
%
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius. ambient for pure water
%     vapor, else dewpoint temperature
%
%Outputs: PPW: partial pressure of water in millibars
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = iceCurve(ATP, temp)
    % Compute the unenchanced partial pressure of water
    %declare the various parameters table 2, curve ei2
    a = 6.1115;
    b = 22.452;
    c = 272.55;

    % compute the partial pressure,
    Ei = a*exp( b*temp/(temp + c) );

    % Compute the enhancement factor

    % declare the various parameters
    A = 3e-4;
    B = 4.18e-6;
    C = 0; %included in case the choice of
    D = 0; % cuves were to change
    E = 0;

    % compute the enhancement factor
    f = 1 + A + ATP*(B + C*(temp + D + E*ATP)^2);

    % Compute the full partial pressure
    PPW = Ei*f;
end
```

2. ExampleEnvironment Class

```
classdef ExampleEnvironment < Geolocation.Environment
   %ExampleEnvironment Provides a set of predefined example
   %environments
   %   The ExampleEnvironment Class provides a set of
   %   predefined example environments which may be selected
   %   by the user in lieu of specifying individual
   %   environmental variables.
   %
   %Example environments include (specify in input Env). . .
   %  MontereySummer
   %  DCWinter
   %  MiamiSummer
   %
   %Author: J. Q. McClintic
   %Date: 20 AUG 2012


   properties (Constant)
   end

   methods
      function[obj] = ExampleEnvironment(Env)
         %swich on environment name
         switch Env
            case 'MontereySummer'
               P = 1000;
               T = 23;
               D = 15;
               E = 2;
            case 'DCWinter'
               P = 1000;
               T = 10;
               D = 5;
               E = 2;
            case 'MiamiSummer'
               P = 1000;
               T = 40;
               D = 35;
               E = 2;
            otherwise
               error(['Specified Example Environment',...
                  ' is not supported.'])
         end%end switch

         obj = obj@Geolocation.Environment(P, T, D, E);
      end
   end
end
```

B. NETWORK CLASS FAMILY

```
classdef Network<handle
   %NETWORK Describes the 802.16 Network configuration
   %   Provides data and network functionality to simulate
   %   the relevant features of an IEEE 802.16 network for
   %   conducting Geolocation simulations.
   %Network(towers,BW,Nused,G): creates an shell network to
   %   hold the network description.
   %   towers: number of towers in the network
   %   BW: bandwidth in Megahertz (MHz)--must be >=1 per
   %      802.16 8.4.1
   %   Nused: number of subcarriers used including DC
   %      subcarrier as defined in 8.4.2.3, 8.4.1
   %   G: ratio of CP time to ``useful" time
   %      (1/32, 1/16, 1/8, 1/4)
   %placeTower(obj,location,n): places a tower
   %   obj: which network place tower in
   %   location: where to put the tower
   %   n: which tower to place
   %randomTowers(obj,dispersion,seperation): generates
   %      random towers for the network
   %   obj: the network to place random towers in
   %   dispersion: 1(x)3 vector of maximum dispersion in
   %      each direction
   %   seperation: minimum seperation in meters between
   %     towers.

   properties (SetAccess = immutable)
      nTowers%number of towers in the network
      BW = 1; %nominal channel bandwidth in MHz
      Nused = 2048; %number of subcarriers including the DC
                    % subcarrier
      G = 1/4; %ratio of CP time to ``useful" time.
   end%end of properties

   properties (SetAccess = private)
      towers%an M (x) 3 matrix of tower locations
      detectThreshold = -90;%detection threshold of a
                            % recieved signal in dBm
   end

   properties(Dependent)
      n%sampling factor--computed as part of initialization
      % or upon change to a relevant parameter per 8.4.2.3
      Nfft%FFT size of the OFDMA
      Fs%sampling frequency in Hertz
```

```
      deltaF%subcarrier spacing in Hertz
      Tb%useful symbol time in seconds
      Tg%CP time in seconds
      Ts%OFDMA symbol time in seconds
      Tsamp%sampling time in seconds
      ps%physical slot length in seconds
      tau%timing adjust unit in seconds
      refTimeTol%reference timing tolerance
      ssFreqTol%the subscriber station maximum center
      %frequency error tollerance
   end

   methods
      % constructors
      function[obj] =  Network(towers, BW, Nused, G)
         %validate attributes
         validateattributes(towers,{'numeric'},...
            {'nonempty','positive', 'scalar'},'','towers')
         validateattributes(BW, {'numeric'},...
            {'nonempty','positive','scalar','>=',1},'','BW')
         validateattributes(Nused,{'numeric'},...
            {'nonempty','positive','scalar','<=',2048},...
            '','Nused')
         validateattributes(G,{'numeric'},...
            {'nonempty','positive','scalar','<',1,'>',0},...
            '','G')
         %check the value of G per 8.4.2.3
         if (G==1/32)||(G==1/16)||(G ==1/8)||(G==1/4)
            obj.G = G;
         else%throw error
            error('G may only be 1/32, 1/16, 1/8, or 1/4.')
         end%end if

         %store initial inputs
         obj.towers = zeros(towers, 3);
         obj.nTowers = towers;
         obj.BW = BW;
         obj.Nused = Nused;
         obj.G = G;
      end%end of constructor

      % setters
```

```
function[] = placeTower(obj, location, n)
   if (n<=length(obj.towers(:,1)))&&(n>=1)%if number
      %   makes sense, place tower
      obj.towers(n, :) = location;
   else%issue error message if tower number is out of
      %bounds
      error([...
         'tower number must be between one and ',...
         num2str(length(obj.towers(:,1)))])
   end
end;%end placeTower


function[] = randomTowers(obj, dispersion, seperation)
   %determine the number of partions of the region for
   %the purposes of placing towers. Use of the common
   %log ensures that the number of partions stays
   %reasonable as the size of the simulation space
   %grows.
   xBreaks = floor(log10(dispersion(1)))+1;
   yBreaks = floor(log10(dispersion(2)))+1;
   %now compute the partition lines in each direction
   temp = 2*dispersion(1)/xBreaks;
   xParts = -dispersion(1):temp:dispersion(1);
   temp = 2*dispersion(1)/yBreaks;
   yParts = -dispersion(1):temp:dispersion(1);
   %compute the number of towers to place in each
   %partition
   minTowers = floor(obj.nTowers/(xBreaks*yBreaks));
   %if at least one tower goes in every block, place
   %that many towers in each of those blocks
   halfSep = seperation/2;
   if minTowers >=1
      zone = 0;%counter for region
      block = zeros(minTowers, 3);%holds towers for
        % this block
      temp = [0 0];%holds the position of a tower
      for x = 1:1:xBreaks
         for y = 1:1:yBreaks
            %compute the lower left and upper right
            %corner positions
            llcx = xParts(x)+halfSep;
```

```
llcy = yParts(y)+halfSep;
urcx = xParts(x+1)-halfSep;
urcy = yParts(y+1)-halfSep;
%make the first tower in the block
block(1, 1) = randi([llcx, urcx], 1, 1);
block(1, 2) = randi([llcy, urcy], 1, 1);
%set the counter for the number of towers
%in the block to one
count = 1;
%while there are fewer towers in the block
%than the minimum number of towers, add
%towers to to block which are at least
%seperation away from each tower already
%in the block.
while count < minTowers
   %make a place to hold distances
   dists = zeros(1, count);
   %make a random tower
   temp(1) = randi([llcx, urcx], 1, 1);
   temp(2) = randi([llcy, urcy], 1, 1);
   %for each tower already in the block,
   %check the distance to this new tower
   for t = 1:1:count
      diff = block(t,1:2) - temp;
      dists(t) = sqrt(dot(diff,diff));
   end%end loop over the towers in
      % the block
   %if the minimum distance is acceptable,
   %place the tower
   if min(dists)>seperation
      %increment count
      count = count+1;
      %add tower to block
      block(count, 1:2) = temp;
   end%end if to add tower to block
end%end while loop to place towers in the
   %block
%add the z-component to the tower
%positions
tempZ = randi([-dispersion(3)...
   dispersion(3)], minTowers, 1);
block(:,3) = tempZ;
```

```
            %push the towers to the object’s tower
            %list
            obj.towers((1+zone*minTowers):...
               (1+zone)*minTowers,:)...
               = block;
            zone = zone+1;
         end%end for each y
      end%end for each x
   end%end if statement (minTowers > 0)
   %now place any additional towers into blocks at
   %random
   remaining = ...
      obj.nTowers - minTowers*xBreaks*yBreaks;
   block = [0 0 0];
   while remaining > 0;
      %pick a block at random
      tempX = randi([1 xBreaks],1,1);
      tempY = randi([1 yBreaks],1,1);
      llcx = xParts(tempX)+halfSep;
      llcy = yParts(tempY)+halfSep;
      urcx = xParts(tempX+1)-halfSep;
      urcy = yParts(tempY+1)-halfSep;
      %randomly generate a tower in that block
      block(1) = randi([llcx, urcx], 1, 1);
      block(2) = randi([llcy, urcy], 1, 1);
      %check if the seperation rule is obeyed
      temp = (obj.towers(:,1) - block(1)).^2;
      temp = temp + (obj.towers(:,2) - block(2)).^2;
      temp = max(sqrt(temp));
      %if it is, then add it to the object’s list of
      %towers and decrement remaining
      total = minTowers*xBreaks*yBreaks;
      if temp > seperation
         block(3) = randi([-dispersion(3)...
            dispersion(3)],1,1);
         obj.towers(total + remaining,:) = block;
         remaining = remaining - 1;
      end%end if statement
   end%end for loop to place remaining towers
end;%end randomTowers

%get.n--set sampling factor per 8.4.2.3
```

```
%obj: the network to configure
function[n] = get.n(obj)
   if mod(obj.BW, 1.75)==0
      n = 8/7;
   elseif mod(obj.BW, 1.25)==0
      n = 28/25;
   else
      n = 8/7;
   end%end if
end%end get.n

%get.Nfft--set the network FFT size per 8.4.2.4
%obj: the network to configure
function[Nfft] = get.Nfft(obj)
   temp = log2(obj.Nused);
   Nfft = 2^ceil(temp);
end%end get.Nfft

%get.Fs--Set sampling frequency per 8.4.2.4
%obj: the network to configure
function[Fs] = get.Fs(obj)
   Fs = floor(1e6*obj.n*obj.BW/8000)*8000;
end%end get.Fs

%get.deltaF--set the subcarrier spacing per 8.4.2.4
%obj: the network to configure
function[deltaF] = get.deltaF(obj)
   deltaF = obj.Fs/obj.Nfft;
end%end get.deltaF

%get.Tb--set the useful symbol time per 8.4.2.4
%obj: the network to configure
function[Tb] = get.Tb(obj)
   Tb = 1/obj.deltaF;
end%end get.Tb

%get.Tg--set CP time per 8.4.2.4
%obj: the network to configure
function[Tg] = get.Tg(obj)
   Tg = obj.G*obj.Tb;
end%end get.Tg
```

```
%get.Ts--set OFDMA symbol time per 8.4.2.4
%obj: the network to configure
function[Ts] = get.Ts(obj)
   Ts = obj.Tb + obj.Tg;
end%end get.Ts

%get.Tsamp--set the sampling time per 8.4.2.4
%obj: the network to configure
%NOTE: all computations done using values
%internal to obj
function[Tsamp] = get.Tsamp(obj)
   Tsamp = obj.Tb/obj.Nfft;
end%end get.Tsamp

%get.ps--set the length of a physical slot (seconds)
%per 10.3.4.2
%obj: the network to configure
function[ps] = get.ps(obj)
   ps = 4/obj.Fs;
end%end get.Ps

%get.tau--set the of a timing adjust unit (seconds)
%per 10.3.4.3
function[tau] = get.tau(obj)
   tau = 1/obj.Fs;
end%end get.tau

%setRefTimeTol--set the absolute value of the
%reference timing tolerance (seconds) per Table 639
function[refTimeTol] = get.refTimeTol(obj)
   refTimeTol = (obj.Tb/32)/4;
end%end get.refTimeTol

%get.ssFreqTol--set the absolute value of the maximum
%subscriber station center frequency error (seconds)
%per 8.4.15.1
function[ssFreqTol] = get.ssFreqTol(obj)
   ssFreqTol = 0.02*obj.deltaF;
end%end setRefTimeTol

% getters
function[towers] = getTower(obj, index)
```

```
         towers = obj.towers(index, :);
      end%end getTowers
   end%end of methods

end%end of Network Class
```

C. TARGET AND ITS RELATED CLASSES

1. Target Class

```
classdef Target < handle
   %TARGET Contains data about the target(s) in the simulation
   %   contains data structures and methods to completely
   %   decribe the location and behavior of simulated
   %   targets.

   properties (SetAccess = protected)
      time%time stamps of position and velocity in
      %milliseconds from simulation start
      position% N (x) 3 matrix of position of target in
         %meters
      velocity% N (x) 3 matrix of velocity of target
      frequency%holds the transmitted frequency of each
         %target in Hz
      power% transmitted power in dBm
      n%number of targets
   end%end properties

   methods
      %% constructor
      %Target(n)--preallocates and empty target object
      %n: number of targets to generate
      function[obj] = Target(n)
         obj.time = zeros(n,1);
         obj.position = zeros(n,3);
         obj.velocity = zeros(n,3);
         obj.n = n;
         obj.frequency = zeros(n,1);
         obj.power = -10+zeros(n,1);%assume maximum power
         %out as the max power recievable by the BS per the
         %802.16 8.4.14.4.2
```

```
end%end constructor

%% setters
function set.time(obj,value)
   if min(value) < 0
      error('Time must be >= to zero');
   elseif isempty(obj.time)
      obj.time = value;
   elseif length(value) ~= length(obj.time)
      error(['value must be the same',...
         ' size as the original']);
   else
      obj.time = value;
   end
end

function set.position(obj,value)
   [r0, c0] = size(obj.position);
   [r1, c1] = size(value);
   if isempty(obj.position)
      obj.position = value;
   elseif (r0 ~= r1)||(c0 ~= c1)
      error(['value must be the same',...
         ' size as the original']);
   else
      obj.position = value;
   end
end

function set.velocity(obj,value)
   [r0, c0] = size(obj.velocity);
   [r1, c1] = size(value);
   if isempty(obj.velocity)
      obj.velocity = value;
   elseif (r0 ~= r1)||(c0 ~= c1)
      error(['value must be the same',...
         ' size as the original']);
   else
      obj.velocity = value;
   end
end
```

```
function set.frequency(obj,value)
   if min(value) < 0
      error(['Frequency must be greater',...
         ' than or equal to zero']);
   elseif isempty(obj.frequency)
      obj.frequency = value;
   elseif length(value) ~= length(obj.frequency)
      error(['value must be the same',...
         ' size as the original']);
   else
      obj.frequency = value;
   end
end

function set.power(obj,value)
   if isempty(obj.power)
      obj.power = value;
   elseif length(value) ~= length(obj.power)
      error(['value must be the same',...
         ' size as the original']);
   else
      obj.power = value;
   end
end

%% getters
function[n] = get.n(obj)
   n = obj.n;
end%end getTime

function[time] = get.time(obj)
   time = obj.time;
end%end getTime

function[position] = get.position(obj)
   position = obj.position;
end%end getPosition

function[velocity] = get.velocity(obj)
   velocity = obj.velocity;
end%end getVelocity
```

```
      function[position, velocity] =...
            getStateByTime(obj, time)
         % compute appropriate index
         temp = ~logical(obj.time - time);

         % handle the request
         if sum(temp ~= 0)
            index = find(temp, 1, 'first');
            position = obj.position(index, :);
            velocity = obj.velocity(index, :);
         else
            error('No data for specified time')
         end
      end%end getStateByTime

      function[time, position, velocity] = ...
            getStateByIndex(obj, index)
         time = obj.time(index);
         position = obj.position(index, :);
         velocity = obj.velocity(index, :);
      end%end getStateByIndex

   end%end methods

end% Target class
```

2. RandomTarget Class

```
classdef RandomTarget < Geolocation.Target
   %RandomTarget This class extends Target by incorporating
   %methods to generate randomly located targets in the
   %target space.
   %INPUTS to class constructor
   %  n: number of targets to create
   %  dispersion: 1 (x) 3 vector of maximum dispersions in
   %  each direction
   %  speed: maximum absolute value of any velocity vector
   %  component
   %  freqRange: 1 (x) 2 vector of the upper and lower
   %  bounds of the range of center frequencies.
```

```
properties (SetAccess = immutable)
   dispersion%1 (x) 3 vector of maximum dispersions
   %in each direction
   speed%maximum value of any velocity vector component
   freqRange%range of expected frequency values in Hz
end

methods
   % constructor
   %RandomTarget--populates object with targets with
   %random positions and velocities
   %obj: target object to populate
   %n: the number of targets to generate
   %speed: maximum value of any velocity vector component
   %freqRange: range of expected frequency values in Hz
   function[obj] = RandomTarget(n, dispersion, speed,...
         freqRange)
      % generate on object of class Target
      obj = obj@Geolocation.Target(n);

      % validate and store inputs for later use
      validateattributes(dispersion, {'numeric'}, ...
         {'vector', 'numel', 3, 'nonnegative'},...
         '','dispersion');
      obj.dispersion = dispersion;
      validateattributes(speed, {'numeric'}, ...
         {'scalar', 'nonnegative'},'','speed');
      obj.speed = speed;
      validateattributes(freqRange, {'numeric'}, ...
         {'vector', 'numel', 2, 'nonnegative'},...
         '','freqRange');
      obj.freqRange = freqRange;

      % make the time, position, and velocity vectors. I
      % should note that the modification of position
      % works with a very restrictive set of conditions
      % embodied in the setter that should preclude the
      % behavior objserved. If it suddenly fails, then
      % there will need to be a temp variable to hold the
      % new matrix and then overwrite the old position
      % matrix.
      obj.time = zeros(obj.n,1);
```

```
         obj.position(:,1) =...
            randi([-dispersion(1),dispersion(1)],...
            [obj.n,1]);
         obj.position(:,2) =...
            randi([-dispersion(2),dispersion(2)],...
            [obj.n,1]);
         obj.position(:,3) =...
            randi([-dispersion(3),dispersion(3)],...
            [obj.n,1]);
         obj.velocity = randi([-speed,speed],obj.n,3);
         obj.frequency = randi(freqRange, obj.n, 1);
      end%end constructor

   end% end methods

end
```

3. CVFWTarget Class

```
classdef CVFWTarget < Geolocation.Target
   %CVFWTarget Constant Velocity Fixed Waypoint Target
   %   This class builds a target with a constant velocity
   %   and constant targets. Inherits from the
   %   Geolocation.Target class.
   %
   %   CVFWTarget(waypoints, speed, n, frequency)
   %   waypoints: waypoints to drive the target through (at
   %   least two)
   %   speed: speed of the target at all times (>0
   %   meters/second)
   %   n: number of observations to construct. (>0)
   %   frequency: center frequency of the transmission
   %   Computes n target position/velocity/time sets evenly
   %   spaced over the path specified by waypoints
   %
   %   The plotTrack method provides three different plots
   %   of the track. If the version argument is ``1" then a
   %   three-D plot is given. If the version is ``2" then
   %   there is a 2-D plot of the X-Y components and a
   %   seperate plot of the Z component. If version is ``3"
   %   then each component is plotted as a seperate function
%   of time.

properties (SetAccess = protected)
   waypoints
   speed
end

properties (SetAccess = immutable)
   legLength%length of each leg
   legPoints%number of points on each leg
   totalDistance%total distance covered by all legs
   cumLength%cummulative length of the legs
   headings%direction of each leg
   timeUnit%time elapsed between observations
   distUnit%distance covered between observations
   wpTime%time of arrival at each waypoint
end

methods
   %% Constructor
   function[obj] = CVFWTarget(waypoints, speed, n,...
         frequency)
      %call the superclass constructor
      obj = obj@Geolocation.Target(n);

      %store the inputs
      obj.waypoints = waypoints;
      obj.speed = speed;
      obj.frequency = frequency.*ones(obj.n, 1);

      %Figure out the heading and length of each leg
      %(legLength)
      [tR, tC] = size(obj.waypoints);
      obj.headings = zeros(tR - 1, tC);
      for ind = 2:1:tR
         %compute the un-normalized heading vectors
         obj.headings(ind -1 , :) = ...
            obj.waypoints(ind, :) ...
            - obj.waypoints(ind-1, :);
      end%end loop over legs
      obj.legLength = ...
         sqrt(...
   diag(obj.headings*transpose(obj.headings)));

%Compute totalDistance and cumLength of the route.
obj.totalDistance = sum(obj.legLength);
obj.cumLength = cumsum(obj.legLength);

%compute some other odds and ends
obj.distUnit = obj.totalDistance/(obj.n-1);
obj.timeUnit = obj.distUnit/obj.speed;

%Compute the number of waypoints on each leg
%Compute the number of points up the n-th waypoint
temp = floor(obj.cumLength./obj.distUnit);
%store the first value directly as it is unchanged
obj.legPoints(1) = temp(1);
%Remove from that total the number the number of
%points cummulative to the previous waypoint to get
%the number on that leg.
for l = 2:1:length(obj.legLength)
   obj.legPoints(l) = temp(l) - temp(l-1);
end%end loop over the legs

%normalize the length of each leg
obj.headings = diag(1./obj.legLength)*obj.headings;

%compute the observation times
obj.time = ((1:1:obj.n)-1).*obj.timeUnit.*1e3;

%compute the waypoint arrival times (wpTime)
obj.wpTime = obj.cumLength./obj.speed;
obj.wpTime = [0 ; obj.wpTime];

%store the position and velocity of the initial
%observation
obj.position(1,:) = obj.waypoints(1,:);
obj.velocity(1,:) = obj.headings(1,:).*obj.speed;
%initialize the observation number
obs = 2;
%For each leg
for l = 1:1:length(obj.legLength)
   %Set any adjustments for this leg
```

```
        %for each point on this leg
        for p = 1:1:obj.legPoints(l)
           %compute the velocity of the target
           obj.velocity(obs,:) = ...
              obj.speed.*obj.headings(l,:);
           %compute the position of the target
           obj.position(obs,:) =...
              obj.waypoints(l,:) + ...
              (obj.time(obs)./1e3-obj.wpTime(l)).* ...
              obj.velocity(obs,:);
           %increment the observation number
           obs = obs+1;
        end%loop over points on leg
     end%end loop over each leg
  end%end constructor CVFWTarget

  %% Setters
  function set.waypoints(obj, value)
     [r, c] = size(value);
     if isempty(obj.waypoints) && (r >= 2) && (c == 3)
        obj.waypoints = value;
     elseif isempty(obj.waypoints) == false
        error('waypoints have already been set.')
     elseif c ~= 3
        error(['All waypoints must have',...
           ' x, y, and z components'])
     else
        error('At least two waypoint must be supplied.')
     end
  end%end set.waypoints

  function set.speed(obj, value)
     if isempty(obj.speed) && (value > 0)
        obj.speed = value;
     elseif value <=0
        error('speed must be >= 0.')
     elseif isempty(obj.speed) == false
        error('speed has already been set.')
     end
  end%end set.speed

  %% Plot Methods
```

```
% write a plot method for the track TO DO: impliment
% code to check the incoming direction and adjust the
% position of the start and end of track labels so
% they do not overwrite the track line
function[] = plotTrack(obj, version)
   %validate arguments
   validateattributes(version, ...
      {'numeric'},...
      {'integer', 'positive', '>=', 1, '<=', 3},...
      '','version')
   %set up the graphic
   figure()
   switch version
      case 1%3D plot
         plot3(obj.position(:,1), ...
            obj.position(:,2),...
            obj.position(:,3),...
            'Marker', 'o')
         xlabel('X Position (meters)')
         ylabel('Y Position (meters)')
         zlabel('Z Position (meters)')
         text(obj.position(obj.n,1), ...
            obj.position(obj.n,2),...
            obj.position(obj.n,3),...
            'End',...
            'VerticalAlignment', 'Bottom')
         text(obj.position(1,1), ...
            obj.position(1,2),...
            obj.position(1,3),...
            'Start',...
            'VerticalAlignment', 'Bottom')
      case 2%X-Y in 2D and seperate Z
         subplot(2, 1, 1)
         plot(obj.position(:,1), obj.position(:,2),...
            'Marker', 'o')
         xlabel('X Position (meters)')
         ylabel('Y Position (meters)')
         text(obj.position(obj.n,1), ...
            obj.position(obj.n,2),...
            'End',...
            'VerticalAlignment', 'Bottom')
         text(obj.position(1,1), ...
                    obj.position(1,2),...
                    'Start',...
                    'VerticalAlignment', 'Bottom')
                 subplot(2, 1, 2)
                 plot(obj.time./1e3, obj.position(:,3),...
                    'Marker', 'o')
                 xlabel('Time (seconds)')
                 ylabel('Z Position (meters)')
              case 3%three seperate component graphs
                 subplot(3,1,1)
                 plot(obj.time./1e3, obj.position(:,1),...
                    'Marker', 'o')
                 xlabel('Time (seconds)')
                 ylabel('X Position (meters)')
                 subplot(3,1,2)
                 plot(obj.time./1e3, obj.position(:,2),...
                    'Marker', 'o')
                 xlabel('Time (seconds)')
                 ylabel('Y Position (meters)')
                 subplot(3,1,3)
                 plot(obj.time./1e3, obj.position(:,3),...
                    'Marker', 'o')
                 xlabel('Time (seconds)')
                 ylabel('Z Position (meters)')
              otherwise%error out
                 error('Please set version as 1, 2, or 3.')
           end%end switch
        end%end the plotTrack method

   end% end methods

end
```

4. CVRWTarget Class

```
classdef CVRWTarget < Geolocation.CVFWTarget
   %CVRWTarget Constant Velocity Random Waypoints Target
   %   This class extends CVFWTarget (Constant Velocity
   %   Fixed Waypoint Target) by allowing the user to
   %   specify the bounds and number of a set of uniformly
   %   randomly positioned waypoints in the track space.
```

```
%
%   INPUTS
%   wpCount: (integer >=2) number of waypoints to
%   generate
%   speed: (positive real) speed of the target in meters
%   per second
%   xbounds: (1 X 2 vector) lower and upper bounds of the
%   x position
%   ybounds: (1 X 2 vector) lower and upper bounds of the
%   y position
%   zbounds: (1 X 2 vector) lower and upper bounds of the
%   z position
%   frequency: (positive real) center frequency of the
%   transmission
%   n: (integer >=1) number of observations to generate

properties (SetAccess = immutable)
wpCount
xbounds
ybounds
zbounds
end

methods
   %% Constructor
   function[obj] = CVRWTarget(wpCount, speed, ...
         xbounds, ybounds, zbounds,...
         n, frequency)

      %validate the inputs
      validateattributes(wpCount, {'numeric'},...
         {'integer','>=',2},...
         '','wpCount')
      validateattributes(xbounds, {'numeric'},...
         {'row', 'ncols', 2, 'integer'},...
         '','xbounds')
      validateattributes(ybounds, {'numeric'},...
         {'row', 'ncols', 2, 'integer'},...
         '','xbounds')
      validateattributes(zbounds, {'numeric'},...
         {'row', 'ncols', 2, 'integer'},...
         '','xbounds')
```

```
            %compute the random waypoints
            tempX = randi(xbounds, wpCount, 1);
            tempY = randi(ybounds, wpCount, 1);
            tempZ = randi(zbounds, wpCount, 1);

            temp = [tempX tempY tempZ];

            %call the superclass constructor
            obj = obj@Geolocation.CVFWTarget(temp, speed,...
               n, frequency);

            %store the inputs
            obj.wpCount = wpCount;
            obj.xbounds = xbounds;
            obj.ybounds = ybounds;
            obj.zbounds = zbounds;

         end%end constructor

         %% Setters

         %% Getters

         %% Plot Methods
      end

   end
```

D. DATA AND ITS RELATED CLASSES

1. Data Class

```
classdef Data
   %DATA provides both abstract properties common to all
   %datasets and a set of functions which may be of use when
   %handling said data. Where applicable, terminology
   %follows the IEEE 802.16 nomenclature.
   %
   %This class cannot actually be insubstantiated but must
   %be used as a superclass for a class which defines the
```

```
%properties in a concete way. These properties are
%n: the number of targets
%timeSigma: the standard deviation of the timing noise
%(assumed to be white Gaussian noise).
%freqSigma: the standard deviation of the frequency noise
%(assumed to be white Gaussian noise).
%timingAdjust: N target by M tower matrix of timing
%adjust data
%frequencyAdjust: N target by M tower matrix of frequency
%adjust data
%frequency: N target by M tower matrix of observed
%frequency
%power: N target by M tower matrix of recieved power
%levels in dBm
%
%Author: J. Q. McClintic
%Data: 27 MAR 2012

properties (SetAccess = private, Abstract)
   n;%number of targets
   timeSigma;%the standard deviation of the timing
   %noise
   freqSigma;%the standard deviation of the frequency
   %noise
   timingAdjust;%N target by M tower matrix of timing
   %adjust data
   frequencyAdjust;%N target by M tower matrix of
   %frequency adjust data
   frequency;%N target by M tower matrix of observed
   %frequency
   power;%N target by M tower matrix of recieved power
   %levels in dBm
end%end properties

methods
   % Constructor
   %Dat--constructor for class Data
   %Env--an object of class Environment
   %Net--an object of class Network
   %Tgt--an object of class Target
   function[obj] = Data()
      %does nothing
```

```
end%end Data

% Getters

%getTowerTA--returns the timing adjust values for all
%targets associated with a given tower
%obj: object of class Data
%n: target number
function[out] = getTowerTA(obj, n)
   out = obj.timingAdjust(:,n);
end%end getTowerTA

%getTowerFrequency--gets the observed frequency of all
%targets at a given tower
%obj: object of class Data
%n: target number
function[out] = getTowerFrequency(obj, n)
   out = obj.frequency(:,n);
end%end getTowerFrequency

%getTowerFA--gets the observed frequency adjust values
%of all targets at a given tower
%obj: object of class Data
%n: target number
function[out] = getTowerFA(obj, n)
   out = obj.frequencyAdjust(:,n);
end%end getTowerFA

%getTargetTA--returns the timing advance values
%associated with a specified target
%obj: object of class Data
%n: target number
function[out] = getTargetTA(obj, n)
   out = obj.timingAdjust(n,:);
end%end getTargetTA

%getTargetFrequency--returns the observed frequency
%from a given target at each tower
%obj: object of class Data
%n: target number
function[out] = getTargetFrequency(obj, n)
   out = obj.frequency(n,:);
```

```
      end%end getTargetFrequency

      %getTargetFA--returns the frequency adjust values from
      %a given target at each tower
      %obj: object of class Data
      %n: target number
      function[out] = getTargetFA(obj, n)
         out = obj.frequencyAdjust(n,:);
      end%end getTargetFA

      %getTargetPower--returns the recieved power of the
      %transmission at each tower
      %obj: object of class Data or derived
      %n: target/observation number
      function[out] = getTargetPower(obj,n)
         out = obj.power(n,:);
      end%end getTargetFA

   end%end methods

   methods(Static)
      %computeTimingAdjust
      %obj: the Data object to manipulate
      %Env--an object of class Environment
      %Net--an object of class Network
      %Tgt--an object of class Target
      function[TA] = computeTimingAdjust(Env, Net, Tgt)
         %variables
         %distance: holds the distances to one tower
         %diff: holds the vector difference between tower
         %location and
         % target location
         tau = Net.tau*Env.c/...
            ppm2n(getRefractivity(Env));%timing adjust
         %units in meters
         TA = zeros(Tgt.n, Net.nTowers);%holds the TA
            %information

         %for each tower
         for k = 1:1:Net.nTowers
            %compute the vector difference
            diff = Tgt.position-repmat(...
         Net.towers(k,:),Tgt.n,1);
      %make the vector of differences using linear
      %algebra
      distance = sqrt(diag(diff*transpose(diff),0));
      %divide each distance by the timing adjust unit
      %in meters
      distance = distance./tau;
      %round the timing adjust units and write to
      %obj.timingAdjust
      TA(:,k) = round(distance);
   end%end loop over towers

end%end computeTimingAdjust

%computeFrequencyAdjust
%Env--an object of class Environment
%Net--an object of class Network
%Tgt--an object of class Target
function[FA] = computeFrequencyAdjust(Env, Net, Tgt)

   %subtract the transmitted frequency from the
   %recieved frequency at each tower to obtain
   %frequency adjust.
   transmitted = ...
      repmat(Tgt.frequency, 1, Net.nTowers);
   FA = round(...
      Geolocation.Data.dopplerShift(Env, Net, Tgt)...
      - transmitted);

end%end computeFrequencyAdjust

%dopplerShift--computes the observed frequencies at
%each reciever
%Env--an object of class Environment
%Net--an object of class Network
%Tgt--an object of class Target
function[F] = dopplerShift(Env, Net, Tgt)
   F = zeros(Tgt.n, Net.nTowers);%holds the recieved
   %frequencies
   vp = ...
      Env.c/ppm2n(getRefractivity(Env));%propagation
      %velocity in the environment
   %for each tower
   for k = 1:1:Net.nTowers
      %compute the normal vectors by first computing
      %the relative position of the targets with
      %respect to the tower
      relPos = Tgt.position - ...
         repmat(Net.towers(k,:),Tgt.n,1);
      %then computing the distance from the tower to
      %each target
      dist = sqrt(diag(relPos*transpose(relPos)));
      %and forming this into a diagonal matrix
      dist = diag(dist);
      %left multiply by the inverse of the distance
      %matrix to obtain the normal vector
      normal = dist\relPos;

      %compute the Line-of-Propagation term
      lop = diag(normal*transpose(Tgt.velocity))./vp;
      %compute the observed frequencies and write to
      %frequency adjust (FA) matrix
      F(:,k) = Tgt.frequency.*(ones(Tgt.n,1) - lop);
   end%end loop over towers

end%dopplerShift

%recievedPower--computes the receieved signal power in
%dBm at each tower.
%This uses only the basic path loss model not the full
%model incorporating factors like coding scheme,
%repetition rate, etc.
%Env--an object of class Environment
%Net--an object of class Network
%Tgt--an object of class Target
%P--the recieved power at each tower in dBm
function[P] = recievedPower(Env, Net, Tgt)
   P = zeros(Tgt.n, Net.nTowers);%holds the recieved
   %power levels

   for tar = 1:1:Tgt.n%for each target
      xmt = Tgt.power(tar);%get the power transmitted
         %by the target
```

```
            for tow = 1:1:Net.nTowers%for each tower compute
                  % the recieved power
               d = getTower(Net,tow) - Tgt.position(tar,:);
               d = sqrt(dot(d,d));
               P(tar,tow) = xmt - ...
                  10*Env.pathLossExponent*log10(d);
            end%end loop over towers
         end%end loop over targets

      end%recievedPower

   end%end static methods

end%end Data class
```

2. SimulatedData Class

```
classdef SimulatedData < Geolocation.Data
   %SimulatedData: A class which provides a masked version
   %of the the dataset appropriate to simulate intermittent
   %data availability of data from passive observation of,
   %for instance, ranging messages.
   %
   %Inputs:
   %Env: an object of class Environment or derived
   %Net: an object of class Network or derived
   %Tgt: an object of class Target or derived
   %
   %NOTE: due to a quirk of MATLAB, if the

   properties (SetAccess = private)
      masked = false;
      mask;
      timeSigma;%the standard deviation of the timing noise
      freqSigma;%the standard deviation of the frequency
         % noise
      n;%number of targets/observations of a target
      timingAdjust;%N target by M tower matrix of timing
         % adjust data
      frequencyAdjust;%N target by M tower matrix of
         %frequency adjust data
      frequency;%N target by M tower matrix of observed
         % frequency
      power;%N target by M tower matrix of recieved power
         % levels in dBm
   end

   methods
      function[obj] = SimulatedData(Env, Net, Tgt, mask)
         validateattributes(mask, {'logical'}, {'nonempty'})

         % set up the underlying data class object there is
         % no call to the superclass constructor because no
         % properties are defined in the superclass, only
         % methods.

         % Set up the underlying true data.
         % Populate timingAdjust data
         obj.timingAdjust = ...
            obj.computeTimingAdjust(Env, Net, Tgt);

         % Populate frequencyAdjust data
         obj.frequencyAdjust = ...
            obj.computeFrequencyAdjust(Env, Net, Tgt);

         % Populate the frequency data
         obj.frequency = obj.dopplerShift(Env, Net, Tgt);

         % Populate the power data
         obj.power = obj.recievedPower(Env, Net, Tgt);

         %compute and populate the number of targets
         obj.n = length(obj.timingAdjust(:,1));

         %handle the case where masking is requested
         if mask == true
            %set the mask flag
            obj.masked = true;
            %set up a default mask--no tower hears the any
            %target at
            % any time
            obj.mask = NaN(obj.n, Net.nTowers);
            %set the number of towers for the initial fix
            hears = 7;
            %set up the mask
            for ind = 1:1:obj.n%each observation
               %--determine which towers ``hear" the target
               %--determine which ones interact with the
               %target
               %--set 1 in all elements of the mask
               %corresponding to towers which interact with
               %the target
               %--retain NaN otherwise
               %this works by determining which entries in
               %the mask should be changed to 1 from NaN.
               pow = getTargetPower(obj,ind);
               temp = sort(pow, 'descend');
               thresh = 0.5*(temp(hears)+temp(hears+1));
               for k = 1:1:Net.nTowers%for each tower
                  if pow(k)>thresh%if sufficient recieved
                        %signal
                     obj.mask(ind,k) = 1;%reset the mask
                        % value
                  end%end if
               end%for each tower

               %determine the number of towers which hear
               %the target
               hears = unidrnd(min(Net.nTowers, 7),1,1);
            end%end for over observations

            obj.timingAdjust = obj.timingAdjust.*obj.mask;
            obj.frequencyAdjust = ...
               obj.frequencyAdjust.*obj.mask;
            obj.frequency = obj.frequency.*obj.mask;
            obj.power = obj.power.*obj.mask;
         end%end handling the case where masking is
            %requested
      end%end constructor

      % Getters
      function[out] = get.timingAdjust(obj)
         out = obj.timingAdjust;
      end
```

```
      function[out] = get.frequencyAdjust(obj)
         out = obj.frequencyAdjust;
      end

      function[out] = get.frequency(obj)
         out = obj.frequency;
      end

      function[out] = get.power(obj)
         out = obj.power;
      end
   end
end
```

3. UserData Class

```
classdef UserData < Geolocation.Data
   %UserData Provides an interface for inputting
   %user-collected data for analysis.
   %   FUNCTIONS
   %
   %   UserData(n, towers): constructor
   %   n: number of observations
   %   towers: vector of tower indices, positive integers
   %
   %   obj = inputData(obj, n, t, value, type): interface
   %   obj: object of class UserData
   %   n: observation number to set
   %   t: tower index associated with observed value
   %   value: an observation-type appropriate measurement in
   %   units defined per Data class definition
   %   type: type of measurement to set.
   %   NOTE: in order to keep the inputted data, the old
   %   UserData-class object must be overwritten because
   %   Data does not inherit from the value class.
   %
   %   The timeSigma and freqSigma properties are listed but
   %   not implimented because they're not currently used in
   %   the code base. This class is simple to extend to the
   %   case where these properties should be set.
```

```
properties(SetAccess = private)
   n;%number of targets
   timeSigma;%the standard deviation of the timing
   %noise
   freqSigma;%the standard deviation of the frequency
   %noise
   timingAdjust;%N target by M tower matrix of timing
   %adjust data
   frequencyAdjust;%N target by M tower matrix of
   %frequency adjust data
   frequency;%N target by M tower matrix of observed
   %frequency
   power;%N target by M tower matrix of recieved power
   %levels in dBm
   towers;%list of towers indices matching those in the
          % corresponding Network-class object
end%end properties

methods
   %Constructor
   function[obj] = UserData(n, towers)
      %validate attributes
      validateattributes(n, {'numeric'}, ...
         {'numel',1,'>=',1, 'nonempty'},'','n')
      validateattributes(towers, {'numeric'}, ...
         {'vector', 'nonempty', ...
         'integer', 'positive'},...
         '','towers')
      %check that the tower serials are unique
      temp = true;%holds whether or not to continue
      t = 1;
      %while temp is true and we have not yet checked
      %each tower
      while (temp == true) && (t <= length(towers))
         if length(find(towers==towers(t)))==1
            %check next by incrementing t
            t = t + 1;
         else
            temp = false;%stop checking
            %output error
            error(['Tower index ',num2str(towers(t)),...
               ' is not unique.'])
         end %end if
      end%end while
      %store the inputs once validated
      obj.n = n;
      obj.towers = sort(towers, 'ascend');

      %set up the data storage
      obj.timingAdjust = ...
         zeros(obj.n, length(obj.towers));
      obj.frequencyAdjust = ...
         zeros(obj.n, length(obj.towers));
      obj.frequency = ...
         zeros(obj.n, length(obj.towers));
      obj.power = ...
         zeros(obj.n, length(obj.towers));
   end%end constructor

   %input a data value
   function[obj] = inputData(obj, n, t, value, type)
      %validate attributes
      validateattributes(obj, ...
         {'Geolocation.UserData'},...
         {'nonempty'}, '','obj')
      validateattributes(n, {'numeric'},...
         {'nonempty', 'integer', ...
         'positive', 'numel', 1, '<=' obj.n}, '', 'n')
      %t (the tower to set data for) is checked by
      %finding it in the towers list of obj
      if length(find(obj.towers==t))==1
         %determine the column to set
         col = find(obj.towers == t);
      else
         %output error
         error(['Tower serial number ',num2str(t),...
            ' is unrecognized'])
      end %end check input t.
      %attributes of value are checked based on the type
      %that the type is acceptable is checked by the
      %switch statment.

      %switch on type and set value
      switch type
```

```
            case 'timingAdjust'
               validateattributes(value, {'numeric'},...
                  {'nonempty', 'positive',...
                  'integer', 'scalar'}, ...
                  '', 'value')
               obj.timingAdjust(n,col) = value;
            case 'frequencyAdjust'
               validateattributes(value, {'numeric'},...
                  {'nonempty', 'integer', 'scalar'},...
                  '', 'value')
               obj.frequencyAdjust(n,col) = value;
            case 'frequency'
               validateattributes(value, {'numeric'},...
                  {'nonempty', 'scalar'}, '', 'value')
               obj.frequency(n,col) = value;
            case 'power'
               validateattributes(value, {'numeric'},...
                  {'nonempty', 'scalar'}, '', 'value')
               obj.power(n,col) = value;
            otherwise
               error('Unrecognized data type')
         end
      end%end inputData
   end%end methods

end%end UserData class
```

E. THE ANALYSIS SUBPACKAGE

1. Doppler

```
classdef Doppler
   %Doppler (Abstract) Provides the common components to all
   %the classes which use the Doppler equation to compute
   %estimated velocity vectors
   %   This abstract class provides the common components of
   %   all classes which use the Doppler equation to compute
   %   estimated velocity vectors. It has abstract
   %   properties
   %   velocity: holds the estimated velocity vectors
   %   N: the assumed refractivity in parts per million
%   T: the number of towers to use in computing the
%   estimate
%
%   The abstract methods defined in this class are
%   [A, b] = constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%   constraints)
%   obj: an object of appropriate class
%   Env: an object of class Environment
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   N: refractivity in ppm
%   tgt: the target to consider
%   constraints: the matrix of tower pairs which
%   represent independant constraints
%   A: the matrix part of the matrix equation
%   b: the vector of constants
%
%   The concrete function defined in this class is
%   [A, b] = constraintEquation(Env, Dat, Nwk, Tgt, N,
%   tgt, i, j, type)
%   i, j: indices of the four towers used in the constraint
%   Net: an object of class Network
%   Dat: an object of class Data
%   Env: an object of class Env
%   N: the assumed refractivity in ppm
%   tgt: target number
%   type: use the recieved frequency (RF) at the towers
%   or the frequency adjust (FA) value.
%   A--row of the matrix
%   b--entry in the column vector
properties(SetAccess = private, Abstract)
   velocity
   N
   T
end%end abstract properties

methods(Abstract)
   [A, b] = constraintMatrix(obj, Env, Nwk, Dat, N, ...
      tgt, constraints)
end%end abstract methods

methods(Abstract, Static)
```

```
    [out] = dopplerConstraintGraph(Nwk, Dat, T, tgt)
end%end abstract, static methods

methods(Static)
   function[A, b] = constraintEquation(Env, Dat, Nwk,...
         Tgt, N, tgt,...
         i, j, type)
      %validate arguments
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(Tgt, {'Geolocation.Target'},...
         {'nonempty'},...
         '','Tgt')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'','N')
      validateattributes(tgt, {'numeric'},...
         {'nonempty', 'integer','positive',...
         '<=',Dat.n},'','tgt')
      validateattributes(i, {'numeric'},...
         {'nonempty','scalar','positive','integer',...
         '<=',Nwk.nTowers})
      validateattributes(i, {'numeric'},...
         {'nonempty','scalar','positive','integer',...
         '<=',Nwk.nTowers})
      if i == j
         error('i cannot equal j')
      end

      %get the target frequency information
      switch type
         case 'FA'
            fData = getTargetFA(Dat, tgt);
         case 'RF'
            fData = getTargetFrequency(Dat, tgt);
            otherwise
               error('type must either be FA or TA')
         end%switch

         %extract the parts of fData needed for this
         %analysis
         fi = fData(i);%the FA/RF at tower i
         fj = fData(j);%the FA/RF at tower j
         fji = fj - fi;%the difference in frequencies
         %between tower i and j

         %get the timing advance information
         temp = getTargetTA(Dat, tgt).*Nwk.tau;
         ti = temp(i);
         tj = temp(j);

         %compute the speed of light estimate
         vp = Env.c/Env.ppm2n(N);

         %get the center frequency of the target
         ft = Tgt.frequency(tgt);

         %output the left and right sides of the constraint
         A = (ft/vp^2).*...
            (getTower(Nwk,j)./tj - getTower(Nwk,i)./ti);
         b = fji;
      end
   end%end static methods
end
```

2. Doppler4

```
classdef Doppler4 < Geolocation.Analysis.Doppler
   %Doppler4 Computes the estimated three-D velocity vector of a target
   %   This class provides methods to compute estimated
   %   velocity vectors for targets given their timing
   %   advance values are known or estimated and the center
   %   frequency of the transmitter is known or an unbiased
   %   estimate of same.
   %   velocity: N (x) 3 matrix of velocity estimates
   properties (SetAccess = private)
   velocity
   N
   T = 4
   type
end%end properties

methods
   % Constructor
   %Velocity--sets up an empty velocity argument
   %Dat: an object of class Data
   function[obj] = Doppler4(Dat, Env, Nwk, Tgt, N,...
         T, type)
      %validate input arguments
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(Tgt, {'Geolocation.Target'},...
         {'nonempty'},...
         '','Tgt')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'','N')
      validateattributes(T, {'numeric'}, ...
         {'nonempty', 'scalar', '>=', 4, '<=', 4},'','T')
      validatestring(type, {'FA','RF'},'','type');
      if Nwk.nTowers < T
         error(['Insufficient towers in network',...
            ' to support analysis.'])
      end

      %store the validated inputs
      obj.N = N;
      obj.T = T;
      obj.type = type;

      %initialize the position estimates
   obj.velocity = zeros(Dat.n,3);%holds results
   %of frequency

   %for each target
   for tgt = 1:1:Dat.n
      %find a set of constraints
      constraints = ...
         obj.dopplerConstraintGraph(Nwk, Dat, T, tgt);

      %form the constraint matrix and vector
      [A,b] = constraintMatrix(obj, Env, ...
         Nwk, Dat, Tgt,...
      N, tgt, type, constraints);

      %solve for the velocity estimate
      obj.velocity(tgt,:) = A\b;
   end%loop over targets
end%end Velocity

function[A, b] = constraintMatrix(obj, Env, Nwk,...
      Dat, Tgt,...
      N, tgt, type, constraints)
   %validate input attributes
   validateattributes(obj,...
      {'Geolocation.Analysis.Doppler4'},...
      {'nonempty'},'','obj')
   validateattributes(Nwk, {'Geolocation.Network'},...
      {'nonempty'},...
      '','Nwk')
   validateattributes(Env, ...
      {'Geolocation.Environment'},...
      {'nonempty'},...
      '','Env')
   validateattributes(Dat, {'Geolocation.Data'},...
      {'nonempty'},...
      '','Dat')
   validateattributes(Tgt, {'Geolocation.Target'},...
      {'nonempty'},...
      '','Tgt')
   validateattributes(tgt, {'numeric'},...
      {'nonempty', 'integer','positive',...
      '<=',Dat.n},'','tgt')
   validateattributes(N, {'numeric'}, ...
      {'nonempty', 'scalar'},'','N')
   validatestring(type, {'FA','RF'},'','type');
   validateattributes(constraints,{'numeric'},...
      {'nonempty','positive','integer','ncols',2},...
      '','constraints')
   [rows,~] = size(constraints);
   if rows < 3
      error('Too few constraints provided.')
   end

   %use the first four constraints
   [A1, b1] = ...
      obj.constraintEquation(Env, Dat, Nwk, Tgt, N,...
      tgt, constraints(1,1), constraints(1,2), type);
   [A2, b2] = ...
      obj.constraintEquation(Env, Dat, Nwk, Tgt, N,...
      tgt, constraints(2,1), constraints(2,2), type);
   [A3, b3] = ...
      obj.constraintEquation(Env, Dat, Nwk, Tgt, N,...
      tgt, constraints(3,1), constraints(3,2), type);

   %pack them into a matrix
   A = [A1;A2;A3];
   b = [b1;b2;b3];
end


% Getters
%getVelocity--gets the velocity estimate for a target
%obj: an object of class Velocity
%n: target number
function[out] = getVelocity(obj, n)
   out = obj.velocity(n,:);
end%end getVelocity

%getSpeed--gets the velocity estimate for a target
%obj: an object of class Velocity
%n: target number
function[out] = getSpeed(obj, n)
   out = sqrt(dot(obj.velocity(n), obj.velocity(n)));
end%end getSpeed
```

```
end%end methods

methods(Static)
   function[out] = ...
         dopplerConstraintGraph(Nwk, Dat, T, tgt)
      %validate the arguemtns
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(tgt, {'numeric'}, ...
         {'nonempty', 'scalar', 'positive',...
         '<=',Dat.n},'','tgt')
      validateattributes(T, {'numeric'}, ...
         {'nonempty', 'scalar', '>=', 4, '<=', 4},'','T')
      if Nwk.nTowers < T
         error(['Insufficient towers in',...
            ' network to support analysis.'])
      end

      %Determine the towers will be used for the
      %analysis. Ensure
      power = Dat.power(tgt,:);%get the power recieved
         %at each tower
      temp = sort(power, 'descend');
      %if the number of towers to consider is less than
         % the total number avaliable
      if T < Nwk.nTowers
         temp = 0.5*(temp(T)+temp(T+1));
      else
         temp = min(temp);
      end
      thresh = min(Nwk.detectThreshold, temp);

      %delete all of the towers which cannot hear the
      %transmitter based on the detection threshold.
      towers = 1:1:Nwk.nTowers;%start with the set of
         %all towers
      for n = Nwk.nTowers:-1:1%go backwards to make this
            %strategy work
            if power(n)<thresh%if insufficient recieved
                  %signal
               towers(n) = [];%delete that tower index
            end%end if
         end%for each tower

         %initialize a Graph object with the remaining
         %towers
         cGraph = Geolocation.Analysis.Graph(towers);
         %compute and store the weights associated with each
         %edge
         tPairs = combnk(towers,2);
         nTP = length(tPairs(:,1));
         for p = 1:1:nTP
            %the weight is 1 so no a priori information is
            %used in selecting constraint equations
            w = 1;
            %store to the graph
            setEdge(cGraph, tPairs(p,1), tPairs(p,2), w)
         end%end loop over pairs

         %Output the result
         out = findTree(cGraph);
      end
   end%end abstract, static methods
end
```

3. Doppler4A

```
classdef Doppler4A < Geolocation.Analysis.Doppler
   %Doppler4A Computes the estimated three-D velocity vector of a target
   %   This class provides methods to compute estimated
   %   velocity vectors for targets given their timing
   %   advance values are known or estimated and the center
   %   frequency of the transmitter is known or an unbiased
   %   estimate of same. This algorithm uses both constraint
   %   choice and an unweighted Doppler Constraint Graph.
   %
   %   Properties:
   %   velocity: N (x) 3 matrix of velocity estimates
   %   N: the assumed refractivity in ppm
%   T: the number of towers to consider in the solution.
%   type: use frequency adjust information ('FA') or raw
%   recieved frequency ('RF').
%
%   Methods:
%   [obj] = Doppler4A(Dat, Env, Nwk, Tgt, N, T, type)
%   Dat: An object of class Data
%   Env: An object of class Environment
%   Nwk: An object of class Network
%   Tgt: an object of class Target (only used to get the
%   origional center frequency)
%   N: the assumed refractivity
%   T: the number of towers to consider in the solution
%   type: use frequency adjust information ('FA') or raw
%   recieved frequency ('RF').
%   obj: an object of class Doppler4B
%
%   [A, b] = constraintMatrix(obj, Env, Nwk, Dat, Tgt,...
%            N, tgt, type, constraints)
%   obj: an object of class Doppler4A
%   Env: An object of class Environment
%   Nwk: An object of class Network
%   Dat: An object of class Data
%   Tgt: an object of class Target (only used to get the
%   origional center frequency)
%   N: the assumed refractivity
%   tgt: the index number of the target or the
%   observation of the target
%   type: use frequency adjust information ('FA') or raw
%   recieved frequency ('RF').
%   constraints: a Nx2 matrix of the pairs of towers
%   which describe the possible constraints.
%   A: a 3x3 matrix
%   b: a 3x1 column vector
%
%   [out] = dopplerConstraintGraph(Nwk, Dat, T, tgt,
%   type) NOTE: Static
%   Nwk: An object of class Network
%   Dat: An object of class Data
%   T: the number of towers to consider in the solution
%   tgt: the index number of the target or the
%   observation of the target
%   type: use frequency adjust information ('FA') or raw
%   recieved frequency ('RF').

properties (SetAccess = private)
   velocity
   N
   T = 4
   type
end%end properties

methods
   % Constructor
   %Velocity--sets up an empty velocity argument
   %Dat: an object of class Data
   function[obj] = Doppler4A(Dat, Env, Nwk, Tgt, N,...
         T, type)
      %validate input arguments
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, ...
         {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(Tgt, ...
         {'Geolocation.Target'},...
         {'nonempty'},...
         '','Tgt')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'','N')
      validateattributes(T, {'numeric'}, ...
         {'nonempty', 'scalar', '>=', 4},'','T')
      validatestring(type, {'FA','RF'},'','type');
      if Nwk.nTowers < T
         error(['Insufficient towers in',...
            ' network to support analysis.'])
      end
      %store the validated inputs
      obj.N = N;
      obj.T = T;
      obj.type = type;

      %initialize the position estimates
      obj.velocity = zeros(Dat.n,3);%holds results of
      %frequency

      %for each target
      for tgt = 1:1:Dat.n
         %find a set of constraints
         constraints = ...
            obj.dopplerConstraintGraph(Nwk, Dat, T, tgt);

         %form the constraint matrix and vector
         [A,b] = constraintMatrix(obj, ...
            Env, Nwk, Dat, Tgt,...
         N, tgt, type, constraints);

         %solve for the velocity estimate
         obj.velocity(tgt,:) = A\b;
      end%loop over targets
   end%end Velocity

   function[A, b] = constraintMatrix(obj, Env, Nwk,...
         Dat, Tgt,...
         N, tgt, type, constraints)
      %validate input attributes
      validateattributes(obj,...
         {'Geolocation.Analysis.Doppler4A'},...
         {'nonempty'},'','obj')
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
```

```
validateattributes(Tgt, {'Geolocation.Target'},...
   {'nonempty'},...
   '','Tgt')
validateattributes(tgt, {'numeric'},...
   {'nonempty', 'integer','positive',...
   '<=',Dat.n},'','tgt')
validateattributes(N, {'numeric'}, ...
   {'nonempty', 'scalar'},'','N')
validatestring(type, {'FA','RF'},'','type');
validateattributes(constraints,{'numeric'},...
   {'nonempty','positive','integer','ncols',2},...
   '','constraints')
[rows,~] = size(constraints);
if rows < 3
   error('Too few constraints provided.')
end

%Initialize A and B
A = zeros(3,3);
b = zeros(3,1);

%compute the set of rows of A
tempA = zeros(length(constraints(:,1)),3);
tempB = zeros(length(constraints(:,1)),1);
for c = 1:1:length(constraints(:,1))
   [tempA(c,:),tempB(c,1)] = ...
      obj.constraintEquation(...
   Env, Dat, Nwk, Tgt, N, tgt,...
   constraints(c,1), constraints(c,2), obj.type);
end%end loop over the possible constraints

%pick the two most orthogonal rows
temp = combnk(1:1:length(constraints(:,1)),2);
tempIP = ...
   zeros(nchoosek(length(constraints(:,1)),2),1);
for c = 1:1:length(temp(:,1))
   tempIP(c) = ...
      abs(dot(tempA(temp(c,1),:),...
      tempA(temp(c,2),:)));
end%end loop over the pairwise combinations of
   %constraints
[~,tempIP] = min(tempIP);%which pair has the
```

```
            %lowest dot product
         tempIP = temp(tempIP,:);%get constraints are most
            %orthogonal

         %pack these rows into the first two rows of A,
         %store the corresponding parts of b
         A(1,:) = tempA(tempIP(1),:);
         A(2,:) = tempA(tempIP(2),:);
         b(1) = tempB(tempIP(1));
         b(2) = tempB(tempIP(2));

         %pick the vector which is most orthogonal to these
         %first two
         tempC = cross(A(1,:), A(2,:));%the vector
            %orthogonal to the first two
         tempCP = zeros(length(constraints(1,:)),1);
         for c = 1:1:length(constraints(:,1))
            tempCP(c,1) = abs(dot(tempA(c,:),tempC));
         end%test each constraint. I choose to retest the
            %two I already have because I know they will be
            %zero
         [~,tempCP] = max(tempCP);%figure out which one is
            %most orthogonal
         A(3,:) = tempA(tempCP,:);%store this vector as the
            %third row of A
         b(3) = tempB(tempCP);%store the entry of b vector
      end
   end

   methods(Static)
      function[out] = dopplerConstraintGraph(Nwk, Dat, T,...
            tgt)
         %validate the arguemtns
         validateattributes(Nwk, {'Geolocation.Network'},...
            {'nonempty'},...
            '','Nwk')
         validateattributes(Dat, {'Geolocation.Data'},...
            {'nonempty'},...
            '','Dat')
         validateattributes(tgt, {'numeric'}, ...
            {'nonempty', 'scalar', 'positive',...
            '<=',Dat.n},'','tgt')
```

```
validateattributes(T, {'numeric'}, ...
   {'nonempty', 'scalar', '>=', 4},'','T')
if Nwk.nTowers < T
   error(['Insufficient towers in',...
      ' network to support analysis.'])
end

%Determine the towers will be used for the
%analysis. Ensure
power = Dat.power(tgt,:);%get the power recieved
   %at each tower
temp = sort(power, 'descend');
%if the number of towers to consider is less than
%the total number avaliable
if T < Nwk.nTowers
   temp = 0.5*(temp(T)+temp(T+1));
else
   temp = min(temp);
end
thresh = min(Nwk.detectThreshold, temp);

%delete all of the towers which cannot hear the
%transmitter based on the detection threshold.
towers = 1:1:Nwk.nTowers;%start with the set of
   %all towers
for n = Nwk.nTowers:-1:1%go backwards to make this
      %strategy work
   if power(n)<thresh%if insufficient recieved
         %signal
      towers(n) = [];%delete that tower index
   end%end if
end%for each tower

%initialize a Graph object with the remaining
%towers
cGraph = Geolocation.Analysis.Graph(towers);
%compute and store the weights associated with
%each edge
tPairs = combnk(towers,2);
nTP = length(tPairs(:,1));
for p = 1:1:nTP
   %the weight is 1 so no a priori information is
```

```
            %used in selecting constraint equations
            w = 1;
            %store to the graph
            setEdge(cGraph, tPairs(p,1), tPairs(p,2), w)
         end%end loop over pairs

         %Output the result
         out = findTree(cGraph);
      end
   end%end abstract, static methods
end
```

4. Graph

```
classdef Graph < handle
   %Graph A class that provides a definition of and basic
   %operations on a graph.
   %   This class provides properties which enable definion
   %   of a graph describing the (weighted)
   %   relationships--edges--between a set of component
   %   members (vertices).
   %
   %Inputs:
   %   vertex: a vector of numeric serial numbers of the
   %   vertices of the graph. There must be at least two
   %   vertices and their serial numbers must be positive
   %   integers. These are assigned index numbers internally
   %   based on their position in the vector. Vector serial
   %   numbers should be unique. Immutable.
   %
   %   edge: If given in the constructor, an N choose 2
   %   vector of weights for the edges where N is the number
   %   of vertices. A weight of zero is taken as ``no edge".
   %   Private.
   %
   %Properties:
   %   vertex: the list of inputted vertices sorting in
   %   order from least to greatest.
   %   edge: the list of edge weights. Zero means no edge.
   %
   %Methods (Public):
```

```
%   Graph(vertex, edge): constructs an object of class
%   graph. The vertex argument is required, edge is
%   optional. If no edge argument is provided, then the
%   edge property is set to a zero vector.
%
%   setEdge(obj, vertex1, vertex2, weight): a set method
%   which allows the user to specify the weight for a
%   single edge. Useful is the set of edges is sparse.
%
%   getEdge(obj, vertex1, vertex2): a get method which
%   returns the weight of the edge specified by vertex1
%   and vertex2.
%
%   findTree(obj): returns a maximum weight spanning tree
%   on the graph specified.
%
%
%Methods (Static):
%   treeMerge(vertex1, vertex2, subtree1, subtree2,
%   vertexList): This function provides a way to merge
%   two trees into a single, larger tree. It takes the
%   arguments:
%   vertex1: the first vertex in the bridging edge
%   vertex2: the second vertex in the bridging edge
%   subtree1: the edge list of the first subtree to merge
%   subtree2: the edge list of the second subtree to
%   merge vertexList: the master list of vertex
%   assignments
%
%   It returns:
%   edgeList: the new edge list for the merged tree.
%   vertexList: the new list of vertex assignments.
%
%   NOTE: This function assumes that the vertices are
%   labelled 1,2,...,N.

properties (SetAccess = immutable)
   vertex
end

properties (SetAccess = private)
   edge
```

```
end

%Internal Properties
properties (SetAccess = immutable, Hidden)
   edgeMap %N choose 2 (x) 4 array. The first two columns
   % are the vertices which define an edge using position
   % labels. The last two columns hold the corresponding
   % serial numbers.
   vertexMap %the N (x) 2 array with the internal
   % indicies in the first column and the vertex labels in
   % the second column.
end

methods
   function[obj] = Graph(vertex, edge)
      %validate the attributes of the vertex argument
      validateattributes(vertex, {'numeric'},...
         {'vector', 'integer', 'positive'},...
         '','vertex')
      %store if vertex passes validation
      obj.vertex = sort(vertex, 'ascend');

      %check the number of arguments
      switch nargin
         case 1
            %initialize the edge vector to the null edge
            %set (no edges)
            obj.edge = zeros(nchoosek(...
               length(obj.vertex),2),1);
         case 2
            %validate the dimensions of the edge argument
            validateattributes(edge, {'numeric'}, ...
               {'numel', nchoosek(...
               length(obj.vertex), 2),'vector'},...
               '','edge')
            %store the edge set provided
            obj.edge = edge;
      end%end switch nargs
      %set up the edge map
      obj.edgeMap(:,1:2) = ...
         flipud(combnk(1:1:length(obj.vertex),2));
      obj.edgeMap(:,3:4) = obj.edgeMap(:,1:2);
      for row = 1:1:nchoosek(length(obj.vertex),2)
         %overwrite the third and forth column entries
         %with the appropriate vertex serial number
         obj.edgeMap(row,3) = ...
            obj.vertex(obj.edgeMap(row,1));
         obj.edgeMap(row,4) = ...
            obj.vertex(obj.edgeMap(row,2));
      end%end for rows

      %set up the vertex map
      obj.vertexMap(:,1) = 1:1:length(obj.vertex);
      obj.vertexMap(:,2) = obj.vertexMap(:,1);
      for row = 1:1:length(obj.vertex)
         obj.vertexMap(row,2) = ...
            obj.vertex(obj.vertexMap(row,1));
      end%end for rows
   end%end Graph

   function[] = setEdge(obj, vertex1, vertex2, weight)
      %validate attributes
      validateattributes(vertex1, {'numeric'},...
         {'scalar', 'positive', 'integer'},'','vertex1')
      validateattributes(vertex2, {'numeric'},...
         {'scalar', 'positive', 'integer'},'','vertex1')
      validateattributes(weight, {'numeric'}, ...
         {'scalar'}, '', 'weight')
      if vertex1 == vertex2
         error('vertex1 cannot equal vertex2.')
      end

      %determine the lower and upper vertex serial number
      v1 = min(vertex1, vertex2);
      v2 = max(vertex1, vertex2);

      %sort the edgeMap so that it is ordered by the
      %vertex serial numbers
      temp = zeros(length(obj.edgeMap(:,1)),2);
      for row = 1:1:length(obj.edgeMap(:,1))
         temp(row,:) = ...
            sort(obj.edgeMap(row,3:4), 'ascend');
      end%for row
      %now search through the temporary edge map to find
      %the indices corresponding to the vertex selected
      [i1,~] = find(temp(:,1) == v1);
      [i2,~] = find(temp(i1,2) == v2 );
      %convert i2 from a relative index to an absolute
      %index. The decrement is an offset required to
      %handle the quirks of counting indices.
      i2 = i1 + i2 - 1;

      %convert the found vertex serial numbers into
      %corresponding serial numbers
      i1 = find(obj.vertex==temp(i1(1),1));
      i2 = find(obj.vertex==temp(i2(1),2));

      %Now find the corresponding entry in the first two
      %columns of the obj.edgeMap property
      v1 = min(i1, i2);
      v2 = max(i1, i2);
      [i1,~] = find(obj.edgeMap(:,1) == v1);
      [i2,~] = find(obj.edgeMap(i1,2) == v2);
      ind = i1 + i2 - 1;

      %Insert the weight into the edge property
      obj.edge(ind(1)) = weight;
   end%end setEdge

   %Write a getEdge function which takes obj, v1, v2 and
   %returns the weight.
   function[out] = getEdge(obj, vertex1, vertex2)
      %validate attributes
      validateattributes(vertex1, {'numeric'},...
         {'scalar', 'positive', 'integer'},'','vertex1')
      validateattributes(vertex2, {'numeric'},...
         {'scalar', 'positive', 'integer'},'','vertex1')
      if vertex1 == vertex2
         error('vertex1 cannot equal vertex2.')
      end

      %determine the lower and upper vertex serial number
      v1 = min(vertex1, vertex2);
      v2 = max(vertex1, vertex2);
      %sort the edgeMap so that it is ordered by the
      %vertex serial numbers
      temp = zeros(length(obj.edgeMap(:,1)),2);
      for row = 1:1:length(obj.edgeMap(:,1))
         temp(row,:) = ...
            sort(obj.edgeMap(row,3:4), 'ascend');
      end%for row

      %now search through the temporary edge map to find
      %the indices corresponding to the vertex selected
      [i1,~] = find(temp(:,1) == v1);
      [i2,~] = find(temp(i1,2) == v2 );
      %convert i2 from a relative index to an absolute
      %index. The decrement is an offset required to
      %handle the quirks of counting indices.
      i2 = i1 + i2 - 1;

      %convert the found vertex serial numbers into
      %corresponding serial numbers
      i1 = find(obj.vertex==temp(i1(1),1));
      i2 = find(obj.vertex==temp(i2(1),2));

      %Now find the corresponding entry in the first two
      %columns of the obj.edgeMap property
      v1 = min(i1, i2);
      v2 = max(i1, i2);
      [i1,~] = find(obj.edgeMap(:,1) == v1);
      [i2,~] = find(obj.edgeMap(i1,2) == v2);
      ind = i1 + i2 - 1;

      %retrieve the edge weight
      out = obj.edge(ind(1));
   end%getEdge

   function[edgeList] = findTree(obj)
      %validate attributes
      validateattributes(obj, ...
         {'Geolocation.Analysis.Graph'},{'nonempty'})

      %set up the vertexAssignment vector
      vertexAssignment = zeros(length(obj.vertex),1);
```

```
%set up the full range of possible subgraphs. The
%first index is the edge number, the second index
%is the vertex number in that edge, and the third
%index is the subgraph number
subGraph = ...
   NaN(length(obj.vertex)-1,2,length(obj.vertex));

%number of edges currently in each subgraph
sgEdgeCount = zeros(length(obj.vertex), 1);

%keep track of the next new subgraph to create
sgCount = 1;% start at one since at least one
% subgraph will be used

%make the augmented edge list. This is a working
%copy.
augEdge(:,1:2) = ...
   flipud(combnk(1:1:length(obj.vertex),2));
augEdge(:,3) = obj.edge;

%make a list of possible bride edges
bridges = zeros(1,3);
numBridges = 0;

%find the highest weight edge. This will be the
%base edge for the tree.
temp = max(obj.edge);%find the max edge weight
temp = find(obj.edge == temp, 1);%find the first
%edge with this weight add it to subgraph 1 and
%assign its vertices in the vertexAssignment vector
subGraph(1,:,1) = augEdge(temp, 1:2);

%increment the counter for the number of edges in
%the first subgraph
sgEdgeCount(1) = 1;

%assign the vertices now in use to the first
%subgraph
vertexAssignment(augEdge(temp, 1)) = sgCount;
vertexAssignment(augEdge(temp, 2)) = sgCount;

%reset the edge weight to -inf. Note that I can't
```

```
%delete the edge because it destroys the indexing
%scheme.
augEdge(temp,3) = -inf;

%while the tree is not yet formed.
while (max(augEdge(:,3)) ~= -inf)&&...
      (max(augEdge(:,3)) ~= 0)
   %find the highest weight edge
   temp = max(augEdge(:,3));
   temp = find(augEdge(:,3) == temp, 1);

   %compute the number of edges in the proposed new
   %edge which are already in use.
   inUse = 0; %assume either is in use
   %check if the first vertex is in use
   if vertexAssignment(augEdge(temp, 1)) ~=0
      inUse = inUse+1;%increment the number of
                      %vertices in use
      %save the SG
      tempSG = vertexAssignment(augEdge(temp, 1));
   end
   %check if the second vertex is in use
   if vertexAssignment(augEdge(temp, 2)) ~=0
      inUse = inUse+1;
      tempSG = vertexAssignment(augEdge(temp, 2));
   end
   %if neither edge is in use, start a new subgraph
   %and increment sgCount
   if inUse == 0
      sgCount = sgCount+1;%set up the new subgraph
      sgEdgeCount(sgCount) =...
         sgEdgeCount(sgCount)+1;%indicate the
      %first edge is now in use;
      subGraph(...
         sgEdgeCount(sgCount), :, sgCount) = ...
         augEdge(temp, 1:2);
      %assign the vertices to this new subgraph
      vertexAssignment(augEdge(temp, 1)) = sgCount;
      vertexAssignment(augEdge(temp, 2)) = sgCount;
      %if has only one shared vertex, add it to
      %that subgraph and increment that subgraph's
      %edge count
   elseif inUse == 1
      sgEdgeCount(tempSG) = sgEdgeCount(tempSG)+1;
      subGraph(sgEdgeCount(tempSG),:,tempSG) =...
         augEdge(temp,1:2);
      vertexAssignment(augEdge(temp, 1)) = tempSG;
      vertexAssignment(augEdge(temp, 2)) = tempSG;
      %if both of its vertices are in use, then add
      %it to bridges and delete it from the augEdge
      %list. Do not keep as a bridge an edge whose
      %vertices are in a single subgraph
   elseif (inUse == 2)&&...
         (vertexAssignment(augEdge(temp, 1)) ~=...
         vertexAssignment(augEdge(temp, 2)))
      %increment the number of bridges
      numBridges = numBridges + 1;
      %add to bridge list
      bridges(numBridges,:) = augEdge(temp,:);
      %if both vertices are in use in the same
      %subgraph
   elseif (inUse == 2)&&...
         (vertexAssignment(augEdge(temp, 1)) ==...
         vertexAssignment(augEdge(temp, 2)))
      %do nothing on purpose
   end
   %once the edge has been handled, reset its
   %weight to -inf.
   augEdge(temp,3) = -inf;
end%end while there are still edges to be checked

%while there is more than one subgraph, merge
%subgraphs together
while sum(sum(isnan(subGraph(:,1:2,1)))) > 0
   %Find the edge in bridges with the highest
   %weight
   bridgeInd = find(bridges(:,3) ==...
      max(bridges(:,3)), 1, 'first');
   %Identify the two subgraphs which it bridges
   v1 = min(bridges(bridgeInd, 1:2));
   v2 = max(bridges(bridgeInd, 1:2));
   sg1 = vertexAssignment(v1);
   sg2 = vertexAssignment(v2);
   %Only use the bridge if the two subgraphs are
```

```
            %different.
            if sg1 ~= sg2
               %call obj.treeMerge to merge the subgraphs
               [tempA, vertexAssignment] = ...
                  Geolocation.Analysis.Graph.treeMerge(...
                  v1, v2, ...
                  subGraph(1:sgEdgeCount(sg1),:,sg1),...
                  subGraph(1:sgEdgeCount(sg2),:,sg2), ...
                  vertexAssignment);
               subGraph(1:1:length(tempA(:,1)),...
                  1:2,min(sg1,sg2)) = tempA;
               %update the number of edges in SG1
               sgEdgeCount(min(sg1,sg2)) =...
                  sgEdgeCount(sg1)+sgEdgeCount(sg2)+1;
               %reset the number of edges assigned to sg2 to
               %zero
               sgEdgeCount(max(sg1,sg2)) = 0;
            end
            %reset the weight of the used bridge to -inf
            bridges(bridgeInd,3) = -inf;
         end%end while merging subgraphs

         %output the edgeList
         edgeList = subGraph(:,1:2,1);
         %convert the elements of the edge list to their
         %equivilent origional vertices
         for r = 1:1:length(edgeList(:,1))
            for c = 1:1:2
               edgeList(r,c) = obj.vertex(edgeList(r,c));
            end
         end
      end%end findTree
   end

   methods(Static)
      function[edgeList, vertexList] = ...
            treeMerge(vertex1, vertex2,...
            subtree1, subtree2, vertexList)
         %validate attributes
         validateattributes(vertex1, {'numeric'}, ...
            {'integer', 'nonempty', 'nonnan'}, '',...
            'vertex1')
            validateattributes(vertex2, {'numeric'}, ...
               {'integer', 'nonempty', 'nonnan'}, '',...
               'vertex2')
            validateattributes(subtree1, {'numeric'}, ...
               {'2d', 'ncols', 2, 'nonnan'}, '', 'subtree1')
            validateattributes(subtree2, {'numeric'}, ...
               {'2d', 'ncols', 2, 'nonnan'}, '', 'subtree2')
            validateattributes(vertexList, {'numeric'},...
               {'vector', 'nonempty'},'', 'vertexList')

            %find the tree to be merged into and the tree to be
            %merged I am choosing to merge down in indices.
            destTree = ...
               min([vertexList(vertex1), vertexList(vertex2)]);
            mergeTree = ...
               max([vertexList(vertex1), vertexList(vertex2)]);
            for ind = find(vertexList == mergeTree)%for each
               %vertex belonging to the tree to be merged,
               %reassign it to the destination tree.
               vertexList(ind) = destTree;
            end
            %add the bridge edge
            subtree1 = [subtree1; vertex1, vertex2];

            %Make the output edgeList
            for row = 1:1:length(subtree1(:,1))
               subtree1(row, :) = sort(subtree1(row,:),...
                  'ascend');
            end%end for row
            for row = 1:1:length(subtree2(:,1))
               subtree2(row, :) = sort(subtree2(row,:),...
                  'ascend');
            end%end for row
            edgeList = sortrows([subtree1 ; subtree2],[1,2]);

         end%end treeMerge
      end%Static functions

   end
```

5. PositionError Class

```
classdef PositionError < handle
   %POSITIONERROR Provides analysis methods for estimated positions
   %   This class, given objects of class Target and any
   %   class which has a position property provides tools to
   %   analyse the structure of the errors.

   properties(SetAccess = private)
      error;%N (x) 3 matrix of raw error vectors
      L2;%L2 norm of the error vectors
      L1;%L1 norm of the error vectors
      Linf;%L0 norm of the error vectors
   end%end properties

   methods
      %% Constructor
      %PositionError--creates and initializes an object of
      %class PositionError
      %Tgt: An object of Class Target containing the true
      %data
      %Est: Any object which contains a property position
      %which contains the estimated positions and a property
      %N the assumed refractivity of the environment
      function[obj] = PositionError(Tgt, Est)
         obj.error = Tgt.position - Est.position;
         obj.L2 = ...
            sqrt(diag(obj.error*transpose(obj.error)));
         obj.L1 =  ...
            transpose(sum(transpose(abs(obj.error))));
         obj.Linf =  ...
            transpose(max(transpose(abs(obj.error))));
      end%end PositionError

      %% Setter


      %% Getter

      %% Plot Methods
      %l2histogram--plots the histogram of the L2 errors
      %obj: an object of class PositionError
      %Est: any object of class which contains the assumed
```

```
%refractivity
%Env: an object of class Environment
%bins: number of bins to use or a vector of bins
%newFig: boolean to indicate whether or not to write
%to a new figure
function[] = l2histogram(obj, Est, Env, bins, newFig)
   %compute the proportions in each bin
   [n, xout] = hist(obj.L2, bins);
   n = n/sum(n);

   %start a new figure as appropriate
   if newFig == true
      figure()
   end

   %create the histogram
   bar(xout, n, 1)
   xlabel('Error (Meters)')
   ylabel('Proportion')
   title({'Histogram of L2 Errors';
      ['True Refractivity: ', ...
      num2str(round(getRefractivity(Env)))];
      ['Assumed Refractivity: ', ...
      num2str(round(Est.N))]})
end%end l2histogram

%l1histogram--plots the histogram of the L2 errors
%obj: an object of class PositionError
%Est: any object of class which contains the assumed
%refractivity
%Env: an object of class Environment
%bins: number of bins to use or a vector of bins
%newFig: boolean to indicate whether or not to write
%to a new figure
function[] = l1histogram(obj, Est, Env, bins, newFig)
   %compute the proportions in each bin
   [n, xout] = hist(obj.L1, bins);
   n = n/sum(n);

   %start a new figure as appropriate
   if newFig == true
      figure()
   end

   %create the histogram
   bar(xout, n, 1)
   xlabel('Error (Meters)')
   ylabel('Proportion')
   title({'Histogram of L1 Errors';
      ['True Refractivity: ', ...
      num2str(round(getRefractivity(Env)))];
      ['Assumed Refractivity: ', ...
      num2str(round(Est.N))]})
end%end l1histogram

%linfhistogram--plots the histogram of the L2 errors
%obj: an object of class PositionError
%Est: any object of class which contains the assumed
%refractivity
%Env: an object of class Environment
%bins: number of bins to use or a vector of bins
%newFig: boolean to indicate whether or not to write
%to a new figure
function[] = linfhistogram(obj, Est, Env, bins,...
      newFig)
   %compute the proportions in each bin
   [n, xout] = hist(obj.Linf, bins);
   n = n/sum(n);

   %start a new figure as appropriate
   if newFig == true
      figure()
   end

   %create the histogram
   bar(xout, n, 1)
   xlabel('Error (Meters)')
   ylabel('Proportion')
   title({'Histogram of L-infinity Errors';
      ['True Refractivity: ', n...
      um2str(round(getRefractivity(Env)))];
      ['Assumed Refractivity: ', ...
      num2str(round(Est.N))]})
end%end linfhistogram
```

```
   end%end methods

end
```

6. TDOA Class

```
classdef TDOA
   %TDOA Provides common functionality and interface for the
   %time differnece of arrival set of classes.
   %   This class defines a set of common parameters and
   %   functions which are either to be implimented across
   %   all TDOA classes.
   %
   %The abstract properties defined in this class are
   %   position--the set of position estimates
   %   N--the refractivity used for the analysis
   %   T--the number of towers to be considered
   %
   %The abstract methods defined in this class are
   %[A, b] = constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
   %constraints)
   %   obj: an object of appropriate class
   %   Env: an object of class Environment
   %   Nwk: an object of class Network
   %   Dat: an object of class Data
   %   N: refractivity in ppm
   %   tgt: the target to consider
   %   constraints: the matrix of tower pairs which
   %   represent independant constraints
   %   A: the matrix part of the matrix equation
   %   b: the vector of constants
   %
   %The concrete function defined in this class is
   %[A, b] = constraintEquation(i,j,k,l, Net, Dat, Env, N,
   %t)
   %i, j, k, l: indices of the four towers used in the
   %constraint
   %Net: an object of class Network
   %Dat: an object of class Data
   %Env: an object of class Env
   %N: the assumed refractivity in ppm
```

```
%t: target number
%A--row of the matrix
%b--entry in the column vector

properties(Abstract, SetAccess = private)
   position
   N
   T
end

methods(Abstract)
   [A, b] = constraintMatrix(obj, ...
      Env, Nwk, Dat, N, tgt, constraints)
end

methods(Abstract, Static)
   [out] = tdoaConstraintGraph(Nwk, Dat, T, tgt)
end

methods(Static)
   function[A, b] = constraintEquation(i,j,k,l, Net,...
         Dat, Env, N, t)
      %get the assumed speed of light and refractive
      %index
      n = ppm2n(N);
      cn = Env.c/n;

      %Get the times
      time = getTargetTA(Dat, t);
      ti = time(i);
      tj = time(j);
      tk = time(k);
      tl = time(l);

      %Make the coefficient
      coef = (tj - ti)/(tl - tk);

      %make A
      %First get the four vectors
      Pi = getTower(Net, i);
      Pj = getTower(Net, j);
      Pk = getTower(Net, k);
```

```
         Pl = getTower(Net, l);
         %Then make the row of A
         A=2*(coef*(Pl - Pk) - (Pj - Pi));

         %make b
         %First make the four magnitudes
         normPi = dot(Pi, Pi);
         normPj = dot(Pj, Pj);
         normPk = dot(Pk, Pk);
         normPl = dot(Pl, Pl);
         %Then make the j-i part
         partji = normPj - normPi - ...
            (Net.tau*cn)^2*(tj^2 - ti^2);
         %Next make the l-k part
         partlk = coef*...
            (normPl - normPk - ...
            (Net.tau*cn)^2*(tl^2 - tk^2));
         %finally, make b
         b=partlk - partji;
      end%end constraintEquation
   end

end
```

7. TDOA5 Class

```
classdef TDOA5 < Geolocation.Analysis.TDOA
   %TDOA5 Time Difference of Arrival 3-D Geolocation using Five Recievers
   %   TDOA5 impliments the closed form algorithm given in
   %   Bakhoum 2006 to solve the passive localization
   %   problem in three dimensions using Time Difference of
   %   Arrival. Properties:
   %     position: an N (x) 3 matrix of target estimated
   %     position
   %     N: the assumed refractivity of the medium
   %     T: the number of towers to use in forming the
   %     estimate
   %
   %   Functions . . .
   %
   %   TDOA5(Env, Net, Dat, N, T, print): returns an object
%   of class TDOA5. Inputs:
%   Env: an object of class Environment
%   Net: an object of class Network
%   Dat: an object of class Data
%   N: Assumed refractivity of the medium (ppm)
%   T: the number of towers to use in forming the
%   estimate
%   print: a Boolean indicating whether or not to print a
%   status message.
%
%   tdoaConstraintGraph(Nwk, Dat, T, tgt): returns an
%   edge list for the TDOA constraint graph. Inputs:
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   T: the number of towers to use in forming the
%   estimate. Must be 5.
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%
%   constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%   constraints): returns a matrix A and a vector b with
%   the linear constraints. Inputs:
%   obj: an object of class TDOA5B
%   Env: an object of class Environment
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   N: Assumed refractivity of the medium (ppm)
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%   constraints: an edge list of the possible constraints
%   from which to choose.

properties(SetAccess = private)
   position%N (x) 3 matrix of target estimated position
   N;%the assumed refractivity
   T = 5;%the number of towers to consider. Fixed at
   %five to conform
   %with Bakhoum's algorithm.
end%End Properties

methods
   %% Constructor
```

```
%TDOA5--Five Reciever Solution in 3D using Bakhoum
%2006's Algorith
%Env: an object of class Environment
%Net: an object of class Network
%Dat: an object of class Data
%N: Assumed refractivity of the medium
function[obj] = TDOA5(Env, Net, Dat, N)
   %validate attributes
   validateattributes(Env, ...
      {'Geolocation.Environment'},...
      {'nonempty'},'','Env')
   validateattributes(Net, {'Geolocation.Network'},...
      {'nonempty'},'','Nwk')
   validateattributes(Dat, {'Geolocation.Data'},...
      {'nonempty'},'','Dat')
   validateattributes(N, {'numeric'}, ...
      {'nonempty', 'scalar'},'TOA','N')

   %convert N from ppm to refractive index
   obj.N = N;

   %preallocate the position matrix
   pos = zeros(Dat.n, 3);

   %for each target compute the position estimate
   for t = 1:1:Dat.n
      %first break the ties using the spanning tree
      %algorithm. Set all weights to one exept of
      %there is a tie.
      pairs = ...
         Geolocation.Analysis. ...
         TDOA5.tdoaConstraintGraph(...
         Net, Dat, obj.T, t);

      %form the constraint matrix
      [A,b] = ...
         constraintMatrix(obj, Env, Net, Dat, N, t, pairs);

      %solve for the unknown position P = A/b
      P0 = A\b;

      %take the transpose of P and write it to the
      %correct row of the position matrix
      pos(t,:) = transpose(P0);
   end%end loop over targets

   %write out
   obj.position = pos;

end%end TDOA5

% Getters
function[out] = getPosition(obj, n)
   out = obj.position(n,:);
end%end getPosition

function[A, b] = ...
      constraintMatrix(obj, Env, Nwk, Dat, N, tgt,...
      constraints)
   %validate inputs
   validateattributes(obj, ...
      {'Geolocation.Analysis.TDOA5'},...
      {'nonempty'},'','obj')
   validateattributes(Env, ...
      {'Geolocation.Environment'},...
      {'nonempty'},'','Env')
   validateattributes(Nwk, {'Geolocation.Network'},...
      {'nonempty'},'','Nwk')
   validateattributes(Dat, {'Geolocation.Data'},...
      {'nonempty'},'','Dat')
   validateattributes(N, {'numeric'}, ...
      {'nonempty', 'scalar'},'TOA','N')
   validateattributes(tgt, {'numeric'},...
      {'nonempty', 'integer','positive',...
      '<=',Dat.n},'','tgt')
   validateattributes(constraints, {'numeric'},...
      {'nonempty','ncols',2,'integer','positive'},...
      '','constraints')

   %use the first five independant constraints
   [A1, b1] = obj.constraintEquation(...
      constraints(2,1), constraints(2,2),...
      constraints(1,1), constraints(1,2),...
      Nwk, Dat, Env, N, tgt);
```

```
        [A2, b2] = obj.constraintEquation(...
           constraints(3,1), constraints(3,2),...
           constraints(1,1), constraints(1,2),...
           Nwk, Dat, Env, N, tgt);
        [A3, b3] = obj.constraintEquation(...
           constraints(4,1), constraints(4,2),...
           constraints(1,1), constraints(1,2),...
           Nwk, Dat, Env, N, tgt);

        %pack the three equations into a matrix
        A = [A1;A2;A3];
        b = [b1;b2;b3];
     end%end constraint matrix

  end%end methods

  methods(Static)
     function[out] = tdoaConstraintGraph(Nwk, Dat, T, tgt)
        %validate inputs
        validateattributes(Nwk, {'Geolocation.Network'},...
           {'nonempty'},'','Nwk')
        validateattributes(Dat, {'Geolocation.Data'},...
           {'nonempty'},'','Dat')
        validateattributes(T, {'numeric'}, ...
           {'integer', '>=', 5, '<=', 5},...
           '', 'T')
        validateattributes(tgt, {'numeric'},...
           {'nonempty', 'integer','positive',...
           '<=',Dat.n},'','tgt')

        if T < Nwk.nTowers
           %Determine the towers will be used for the
           %analysis. Ensure
           power = Dat.power(tgt,:);%get the power
              %recieved at each tower
           temp = sort(power, 'descend');
           temp = 0.5*(temp(T)+temp(T+1));
           thresh = min(Nwk.detectThreshold, temp);

           %delete all of the towers which cannot hear the
           %transmitter based on the detection threshold.
           towers = 1:1:Nwk.nTowers;%start with the set of
```

```
                %all towers
            for n = Nwk.nTowers:-1:1%go backwards to make
                  %this strategy work
               if power(n)<thresh%if insufficient recieved
                     %signal
                  towers(n) = [];%delete that tower index
               end%end if
            end%for each tower
         else%if there are exactly 5 towers in the system
            towers = 1:1:Nwk.nTowers;
         end
         %set of the graph
         graph = Geolocation.Analysis.Graph(towers);
         %compute all possible edges
         edges = combnk(towers,2);
         %get the timing advance values associated with this
         %target
         temp = getTargetTA(Dat, tgt);

         %For each possible edge . . .
         for ind = 1:1:length(edges(:,1))
            %set the weight. If the time difference is not
            %zero, onee
            if temp(edges(ind,1)) ~= temp(edges(ind,2))
               setEdge(graph, edges(ind,1), edges(ind,2),1)
            else%edge weight is zero
               setEdge(graph,edges(ind,1), edges(ind,2),0)
            end%end set weight on edge ind
         end%end loop over possible edges
         out = findTree(graph);%find a tree
      end%end tdoaConstraintGraph
   end%end static methods

end %end TDOA5
```

8. TDOA5A Class

```
classdef TDOA5A < Geolocation.Analysis.TDOA
   %TDOA5A Time Difference of Arrival 3-D Geolocation using
   %Five Recievers
   %   TDOA5A impliments a modified version of the closed
%   form algorithm given in Bakhoum 2006 to solve the
%   passive localization problem in three dimensions
%   using Time Difference of Arrival. The modifications
%   are the incorporation of an unweighted constraint
%   choice algorithm. Properties:
%     position: an N (x) 3 matrix of target estimated
%     position
%     N: the assumed refractivity of the medium
%     T: the number of towers to use in forming the
%     estimate
%
%   Functions . . .
%
%   TDOA5A(Env, Net, Dat, N, T, print): returns an object
%   of class TDOA5B. Inputs:
%   Env: an object of class Environment
%   Net: an object of class Network
%   Dat: an object of class Data
%   N: Assumed refractivity of the medium (ppm)
%   T: the number of towers to use in forming the
%   estimate
%   print: a Boolean indicating whether or not to print a
%   status message.
%
%   getPosition(obj): returns the estimated position of a
%   specified target. Inputs:
%   obj: an object of class TDOA5B
%   n: target number
%
%   tdoaConstraintGraph(Nwk, Dat, T, tgt): returns an
%   edge list for the TDOA constraint graph. Inputs:
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   T: the number of towers to use in forming the
%   estimate (>=5)
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%
%   constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%   constraints): returns a matrix A and a vector b with
%   the linear constraints. Inputs:
%   obj: an object of class TDOA5B
%   Env: an object of class Environment
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   N: Assumed refractivity of the medium (ppm)
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%   constraints: an edge list of the possible constraints
%   from which to choose.

properties(SetAccess = private)
   position%N (x) 3 matrix of target estimated position
   N;%the assumed refractivity
   T = 7;
end%End Properties

methods
   % Constructor
   %TDOA5A--Five Reciever Solution in 3D using Bakhoum
   %2006's Algorith
   %Env: an object of class Environment
   %Net: an object of class Network
   %Dat: an object of class Data
   %N: Assumed refractivity of the medium
   function[obj] = TDOA5A(Env, Net, Dat, N, T, print)
      %validate attributes
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},'','Env')
      validateattributes(Net, {'Geolocation.Network'},...
         {'nonempty'},'','Net')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},'','Dat')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(T, {'numeric'}, ...
         {'integer', '>=', 5, '<=', Net.nTowers},...
         '', 'T')
      validateattributes(print, {'logical'},...
         {'nonempty'},'','print')

      %convert N from ppm to refractive index
      obj.N = N;
      %store T
      obj.T = T;

      %preallocate the position matrix
      obj.position = zeros(Dat.n, 3);

      %for each target compute the position estimate
      for t = 1:1:Dat.n

         %compute a tree from the TDOA constraint ...
         graph in adjacency
         %list form
         tree = ...
            Geolocation.Analysis. ...
            TDOA5A.tdoaConstraintGraph(...
            Net, Dat, obj.T, t);

         %compute the constraint matrices
         [A, b] = constraintMatrix(obj, ...
            Env, Net, Dat, obj.N, t, tree);

         %write out the result
         obj.position(t,:) = A\b;

         %print a status message
         if print == true
            disp(['Target ', num2str(t), ' complete.'])
         end%status message
      end%end loop over targets
   end%end TDOA5
   % Setters

   % Getters
   function[out] = getPosition(obj, n)
      out = obj.position(n,:);
   end%end getPosition

   function [A, b] = constraintMatrix(obj, ...
         Env, Nwk, Dat, N, tgt, constraints)
      %validate inputs
      validateattributes(obj, ...
         {'Geolocation.Analysis.TDOA5A'},...
         {'nonempty'},'','obj')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},'','Env')
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},'','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},'','Dat')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(tgt, {'numeric'},...
         {'nonempty', 'integer','positive',...
         '<=',Dat.n},'','tgt')
      validateattributes(constraints, {'numeric'},...
         {'nonempty','ncols',2,'integer','positive'},...
         '','constraints')

      %Compute all the possible pairs of
      %constraints--that is, pairs of rows of the
      %constraint graph.
      conPairs = combnk(1:1:length(constraints(:,1)),2);
      %compute all the possible constraint equation
      A = zeros(nchoosek(length(conPairs(:,1)),2),3);
      b = A(:,1);
      %for each edge pair (linear tdoa constraint)
      for ind = 1:1:length(conPairs(:,1))
         [A(ind,:),b(ind,1)] = obj.constraintEquation(...
            constraints(conPairs(ind,2),1),...
            constraints(conPairs(ind,2),2),...
            constraints(conPairs(ind,1),1),...
            constraints(conPairs(ind,1),2),...
            Nwk, Dat, Env, N, tgt);
      end
      %find the two most orthogonal
      innerProducts = zeros(length(conPairs(:,1)),1);
      rowPairs = combnk(1:1:length(conPairs(:,1)),2);
      %loop over all the pairs of rows of A
      for ind = 1:1:length(rowPairs(:,1))
         %compute the inner product--this is faster than
         %the cross product.
         innerProducts(ind,1) = abs(dot(A(rowPairs(ind,1),:),...
                      A(rowPairs(ind,2),:)));
                end%loop over row pairs
                [~,whichMin] = min(innerProducts);%find the pair
                   %with the minimum inner product
                A1 = A(rowPairs(whichMin,1),:);
                A2 = A(rowPairs(whichMin,2),:);
                b1 = b(rowPairs(whichMin,1),1);
                b2 = b(rowPairs(whichMin,2),1);
                %compute the vector normal to the first two
                %constraints
                normVT = cross(A1,A2);
                %find the constraint most parallel to the normal
                %vector
                dotProds = zeros(length(A(:,1)),1);
                for ind = 1:1:length(A(:,1))
                   dotProds(ind,1) = dot(normVT,A(ind,:));
                end%
                whichMax = ...
                   find(dotProds == max(dotProds),1,'first');
                A3 = A(whichMax,:);
                b3 = b(whichMax,1);

                %pack the three equations into a matrix
                A = [A1;A2;A3];
                b = [b1;b2;b3];
             end%end constraintMatrix
          end%end methods

          methods(Static)
             function[out] = ...
                   tdoaConstraintGraph(Nwk, Dat, T, tgt)
                %validate inputs
                validateattributes(Nwk, {'Geolocation.Network'},...
                   {'nonempty'},'','Nwk')
                validateattributes(Dat, {'Geolocation.Data'},...
                   {'nonempty'},'','Dat')
                validateattributes(T, {'numeric'}, ...
                   {'integer', '>=', 5, '<=', Nwk.nTowers},...
                   '', 'T')
                validateattributes(tgt, {'numeric'},...
                   {'nonempty', 'integer','positive',...
                   '<=',Dat.n},'','tgt')
```

```
if T < Nwk.nTowers
   %Determine the towers will be used for the
   %analysis. Ensure
   power = Dat.power(tgt,:);%get the power
      %recieved at each tower
   temp = sort(power, 'descend');
   temp = 0.5*(temp(T)+temp(T+1));
   thresh = min(Nwk.detectThreshold, temp);

   %delete all of the towers which cannot hear the
   %transmitter based on the detection threshold.
   towers = 1:1:Nwk.nTowers;%start with the set of
      %all towers
   for n = Nwk.nTowers:-1:1%go backwards to make
         %this strategy work
      if power(n)<thresh%if insufficient recieved
            %signal
         towers(n) = [];%delete that tower index
      end%end if
   end%for each tower
else% if the user wants all towers in the system
      %considered
   towers = 1:1:Nwk.nTowers;%keep all towers
end%if T < Nwk.nTowers

%set of the graph
graph = Geolocation.Analysis.Graph(towers);
%compute all possible edges
edges = combnk(towers,2);
%get the timing advance values associated with this
%target
temp = getTargetTA(Dat, tgt);

%For each possible edge . . .
for ind = 1:1:length(edges(:,1))
   %set the weight. If the time difference is not
   %zero, onee
   if temp(edges(ind,1)) ~= temp(edges(ind,2))
      setEdge(graph, edges(ind,1), edges(ind,2),1)
   else%edge weight is zero
      setEdge(graph,edges(ind,1), edges(ind,2),0)
```

```
            end%end set weight on edge ind
         end%end loop over possible edges
         out = findTree(graph);%find a tree
      end%tdoaConstraintGraph
   end%end static methods
end %end TDOA5A
```

9. TDOA5B Class

```
classdef TDOA5B < Geolocation.Analysis.TDOA
   %TDOA5B Time Difference of Arrival 3-D Geolocation using Five Reciever
   %   TDOA5B impliments a modified version of the closed
   %   form algorithm given in Bakhoum 2006 to solve the
   %   passive localization problem in three dimensions
   %   using Time Difference of Arrival. The modifications
   %   are the incorporation of a weigthed constraint choice
   %   algorithm. Properties:
   %     position: an N (x) 3 matrix of target estimated
   %     position
   %     N: the assumed refractivity of the medium
   %     T: the number of towers to use in forming the
   %     estimate
   %
   %   Functions . . .
   %
   %   TDOA5B(Env, Net, Dat, N, T, print): returns an object
   %   of class TDOA5B. Inputs:
   %   Env: an object of class Environment
   %   Net: an object of class Network
   %   Dat: an object of class Data
   %   N: Assumed refractivity of the medium (ppm)
   %   T: the number of towers to use in forming the
   %   estimate (>=5)
   %   print: a Boolean indicating whether or not to print a
   %   status message.
   %
   %   getPosition(obj): returns the estimated position of a
   %   specified target. Inputs:
   %   obj: an object of class TDOA5B
   %   n: target number
   %
```

```
%   tdoaConstraintGraph(Nwk, Dat, T, tgt): returns an
%   edge list for the TDOA constraint graph. Inputs:
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   T: the number of towers to use in forming the
%   estimate
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%
%   constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%   constraints): returns a matrix A and a vector b with
%   the linear constraints. Inputs:
%   obj: an object of class TDOA5B
%   Env: an object of class Environment
%   Nwk: an object of class Network
%   Dat: an object of class Data
%   N: Assumed refractivity of the medium (ppm)
%   tgt: the target number (or the observation of the
%   target) for which to compute the constraint graph.
%   constraints: an edge list of the possible constraints
%   from which to choose.

properties(SetAccess = private)
   position
   N;
   T = 7;
end%End Properties

methods
   %% Constructor
   %TDOA5B--Five Reciever Solution in 3D using Bakhoum
   %2006's Algorith

   function[obj] = TDOA5B(Env, Net, Dat, N, T, print)
      %validate attributes
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},'','Env')
      validateattributes(Net, ...
         {'Geolocation.Network'},...
         {'nonempty'},'','Net')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},'','Dat')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(T, {'numeric'},...
         {'integer', '>=', 5, '<=', Net.nTowers},...
         '', 'T')
      validateattributes(print, {'logical'},...
         {'nonempty'},'','print')

      %convert N from ppm to refractive index
      obj.N = N;

      %store T
      obj.T = T;

      %preallocate the position matrix
      obj.position = zeros(Dat.n, 3);

      %for each target compute the position estimate
      for t = 1:1:Dat.n

         %compute a tree from the TDOA constraint graph
         %in adjacency list form
         tree = Geolocation.Analysis. ...
            TDOA5B.tdoaConstraintGraph(...
            Net, Dat, obj.T, t);

         %compute the constraint matrices
         [A, b] = constraintMatrix(obj, ...
            Env, Net, Dat, obj.N, t, tree);

         %write out the result
         obj.position(t,:) = A\b;

         %print a status message
         if print == true
            disp(['Target ', num2str(t), ' complete.'])
         end%status message
      end%end loop over targets
   end%end TDOA5
   % Setters
```

```
% Getters
function[out] = getPosition(obj, n)
   out = obj.position(n,:);
end%end getPosition

function [A, b] = constraintMatrix(obj, ...
      Env, Nwk, Dat, N, tgt, constraints)
   %validate inputs
   validateattributes(obj, ...
      {'Geolocation.Analysis.TDOA5B'},...
      {'nonempty'},'','obj')
   validateattributes(Env, ...
      {'Geolocation.Environment'},...
      {'nonempty'},'','Env')
   validateattributes(Nwk, {'Geolocation.Network'},...
      {'nonempty'},'','Nwk')
   validateattributes(Dat, {'Geolocation.Data'},...
      {'nonempty'},'','Dat')
   validateattributes(N, {'numeric'}, ...
      {'nonempty', 'scalar'},'TOA','N')
   validateattributes(tgt, {'numeric'},...
      {'nonempty', 'integer','positive','<=',...
      Dat.n},'','tgt')
   validateattributes(constraints, {'numeric'},...
      {'nonempty','ncols',2,'integer','positive'},...
      '','constraints')

   %Compute all the possible pairs of
   %constraints--that is, pairs of rows of the
   %constraint graph.
   conPairs = combnk(1:1:length(constraints(:,1)),2);
   %compute all the possible constraint equation
   A = zeros(nchoosek(length(conPairs(:,1)),2),3);
   b = A(:,1);
   %for each edge pair (linear tdoa constraint)
   for ind = 1:1:length(conPairs(:,1))
      [A(ind,:),b(ind,1)] = obj.constraintEquation(...
         constraints(conPairs(ind,2),1),...
         constraints(conPairs(ind,2),2),...
         constraints(conPairs(ind,1),1),...
         constraints(conPairs(ind,1),2),...
         Nwk, Dat, Env, N, tgt);
      end
      %find the two most orthogonal
      innerProducts = zeros(length(conPairs(:,1)),1);
      rowPairs = combnk(1:1:length(conPairs(:,1)),2);
      %loop over all the pairs of rows of A
      for ind = 1:1:length(rowPairs(:,1))
         %compute the inner product--this is faster than
         %the cross product.
         innerProducts(ind,1) = abs(dot(A(rowPairs(ind,1),:),...
            A(rowPairs(ind,2),:)));
      end%loop over row pairs
      [~,whichMin] = min(innerProducts);%find the pair
         %with the minimum inner product
      A1 = A(rowPairs(whichMin,1),:);
      A2 = A(rowPairs(whichMin,2),:);
      b1 = b(rowPairs(whichMin,1),1);
      b2 = b(rowPairs(whichMin,2),1);
      %compute the vector normal to the first two
      %constraints
      normVT = cross(A1,A2);
      %find the constraint most parallel to the normal
      %vector
      dotProds = zeros(length(A(:,1)),1);
      for ind = 1:1:length(A(:,1))
         dotProds(ind,1) = dot(normVT,A(ind,:));
      end%
      whichMax = ...
         find(dotProds == max(dotProds),1,'first');
      A3 = A(whichMax,:);
      b3 = b(whichMax,1);

      %pack the three equations into a matrix
      A = [A1;A2;A3];
      b = [b1;b2;b3];
   end%end constraintMatrix
end%end methods

methods(Static)
   function[out] = tdoaConstraintGraph(Nwk, Dat, T, tgt)
      %validate inputs
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},'','Nwk')
```

```
validateattributes(Dat, {'Geolocation.Data'},...
   {'nonempty'},'','Dat')
validateattributes(T, {'numeric'}, ...
   {'integer', '>=', 5, '<=', Nwk.nTowers},...
   '', 'T')
validateattributes(tgt, {'numeric'},...
   {'nonempty', 'integer','positive',...
   '<=',Dat.n},'','tgt')

if T < Nwk.nTowers
   %Determine the towers will be used for the
   %analysis. Ensure
   power = Dat.power(tgt,:);%get the power recieved
      %at each tower
   temp = sort(power, 'descend');
   temp = 0.5*(temp(T)+temp(T+1));
   thresh = min(Nwk.detectThreshold, temp);

   %delete all of the towers which cannot hear the
   %transmitter based on the detection threshold.
   towers = 1:1:Nwk.nTowers;%start with the set of
      %all towers
   for n = Nwk.nTowers:-1:1%go backwards to make
         %this strategy work
      if power(n)<thresh%if insufficient recieved
            %signal
         towers(n) = [];%delete that tower index
      end%end if
   end%for each tower
else% if the user wants all towers in the system
      %considered
   towers = 1:1:Nwk.nTowers;%keep all towers
end%if T < Nwk.nTowers

%set of the graph
graph = Geolocation.Analysis.Graph(towers);
%compute all possible edges
edges = combnk(towers,2);
%get the timing advance values associated with this
%target
temp = getTargetTA(Dat, tgt);
```

```
         %For each possible edge . . .
         for ind = 1:1:length(edges(:,1))
            %set the weight. If the time difference is not
            %zero,
            if temp(edges(ind,1)) ~= temp(edges(ind,2))
               w = abs(temp(edges(ind,1)) - ...
                  temp(edges(ind,2)));
            else%edge weight is zero
               w = 0;
            end%end set weight on edge ind
            setEdge(graph,edges(ind,1), edges(ind,2),w)
         end%end loop over possible edges
         out = findTree(graph);%find a tree
      end%tdoaConstraintGraph
   end%end static methods
end %end TDOA5B
```

10. TOA Class

```
classdef TOA
   %TOA Provides common functionality and variables for all TOA classes
   %   This class provides three properties which are
   %   abstract and two instantiated static functions which
   %   are of use across all the time of arrival algorithm
   %   implimenting classes.
   %
   %The properties are:
   %   position--holds the estimated positions
   %   N--the refractivity used for the estimates
   %   T--the number of towers considered in the analysis.
   %
   %toaConstraintEquation(): a static method which returns
   %two outputs. The first is a vector which is a row of the
   %solution matrix. The second is the constant for the
   %associated vector of constants. Takes arguments
   %  Env: an object of class Geolocation.Environment
   %  Nwk: an object of class Geoloction.Network
   %  Dat: an object of class Geoloction.Data
   %  N: the assumed refractivity in parts per million
   %  tgt: the target number to consider
   %  t1: the first tower to use in constructing the
%  constraint
%  t2: the second tower to use in constructing the
%  constraint

properties(Abstract, SetAccess = immutable)
   position
   N
   T
end

methods(Abstract)
   [A,b] = constraintMatrix(obj, ...
      Env, Nwk, Dat, N, tgt, constraints)
end

methods(Static, Abstract)
   [out] = toaConstraintGraph(Nwk, Dat, T, tgt)
end

methods(Static)
   function[rowA, rowB] = toaConstraintEquation(...
         Env, Nwk, Dat, tgt, N, t1, t2)
      %validate attributes
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(tgt,{'numeric'},...
         {'nonempty', 'scalar', 'positive',...
         '<=',Dat.n},'','tgt')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(t1,{'numeric'},...
         {'nonempty', 'scalar', 'positive',...
         '<=',Nwk.nTowers},'','t1')
      validateattributes(t2,{'numeric'},...
                {'nonempty', 'scalar', 'positive',...
                '<=',Nwk.nTowers},'','t2')

            %extract the positions of towers t1 and t2
            pi = getTower(Nwk, t1);
            pj = getTower(Nwk, t2);

            %create the ranges from each tower to the target
            temp = getTargetTA(Dat, tgt);
            vp = Env.c/Env.ppm2n(N);
            ri = vp*temp(t1)*Nwk.tau;
            rj = vp*temp(t2)*Nwk.tau;

            %solve for the distance from t1 to the center of
            %the constraint circle
            dij = sqrt(dot(pj-pi,pj-pi));
            tempTop = ri^2 - rj^2 + dot(pj-pi,pj-pi);
            tempBot = 2*dij;
            di = tempTop/tempBot;

            %solve for the center of the constraint circle
            pij = pi + (di/dij).*(pj-pi);

            %solve for the orientation of the constraint plane
            nij = (pj-pi)./dij;
            %construct row of A
            rowA = nij;

            %compute the element of B
            rowB = dot(nij,pij);
        end%end toaConstraintEquation
    end
end
```

11. TOA4 Class

```
classdef TOA4 < Geolocation.Analysis.TOA
   %TOA4 Time Of Arrival Position Solver
   %   This class provides an algorithm to solve for the
   %   position of a target in three dimensions given
   %   objects of class Network and Data along with an
%   estimated refractivity. This class cannot handle
%   missing observations.
%
%FUNCTIONS
%TOA4(Nwk, Dat, N, T): The constructor. Always considers
%only the four closest towers by recieved power. Takes
%arguments
%  Nwk: an object of class Geoloction.Network
%  Dat: an object of class Geoloction.Data
%  N: the assumed refractivity in parts per million
%Outputs:
%  position: an Mx3 matrix of estimated positions.
%
%constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%constraints)
%  returns the set of ``best" (in this class, no weights)
%  constraints packed into matrix form for solving
%  position estimation problem. Takes inputs
%  obj: an object of class TOA4
%  Nwk: an object of class Network
%  Env: an object of class Geolocation.Environment
%  Dat: an object which inherits from class Data
%  N: the assumed refractivity in parts per million
%  tgt: the target number to consider
%  constraints: a list of tower pairs which are the
%  possible constraints
%
%toaConstraintGraph(Nwk, Dat, T, tgt): a static method
%which returns a TOA Constraint Graph which is a list of
%the tower pairs to use in forming the final solution.
%Takes arguments
%  Nwk: an object of class Geoloction.Network
%  Dat: an object of class Geoloction.Data
%  N: the assumed refractivity in parts per million
%  T: number of towers to consider when forming the final
%  solution
%  tgt: the target number to consider
%
%toaConstraintEquation(): a static method which returns
%two outputs. The first is a vector which is a row of the
%solution matrix. The second is the constant for the
%associated vector of constants. Takes arguments
```

```
% Env: an object of class Geolocation.Environment
% Nwk: an object of class Geoloction.Network
% Dat: an object of class Geoloction.Data
% N: the assumed refractivity in parts per million
% tgt: the target number to consider
% t1: the first tower to use in constructing the
% constraint
% t2: the second tower to use in constructing the
% constraint

properties(SetAccess = immutable)
   position
   N
   T = 4;
end

methods
   function[obj] = TOA4(Nwk, Dat, Env, N)
      %validate object inputs
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         'TOA','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         'TOA','Dat')

      %record the refractivity
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      obj.N = N;

      %initialize the position matrix
      obj.position = zeros(Dat.n, 3);

      %FOR each target
      for ind = 1:1:Dat.n
         %Select a set of independant constraints
         constraints = obj.toaConstraintGraph(Nwk, ...
            Dat, 4, ind);
         %Select those constraints which maximimze the
         %determinant of the constraint matrix
         [A,b] = constraintMatrix(obj, ...
            Env, Nwk, Dat, N, ind, constraints);

         %Solve the matrix equation and save
         obj.position(ind, :) = A\b;
      end%end FOR each target
   end%End constructor

   function[A,b] = constraintMatrix(obj, ...
         Env, Nwk, Dat, N, tgt, constraints)
      %validate attributes
      validateattributes(obj, ...
         {'Geolocation.Analysis.TOA4'},...
         {'nonempty'},...
         '','obj')
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(tgt, {'numeric'},...
         {'nonempty', 'integer','positive',...
         '<=',Dat.n},'','tgt')
      validateattributes(constraints, {'numeric'},...
         {'nonempty','ncols',2,'integer','positive'},...
         '','constraints')
      %Initialize A and B
      A = zeros(3,3);
      b = zeros(3,1);

      %compute the set of rows of A
      tempA = zeros(length(constraints(:,1)),3);
      tempB = zeros(length(constraints(:,1)),1);
      for c = 1:1:length(constraints(:,1))
         [tempA(c,:),tempB(c,1)] = ...
            obj.toaConstraintEquation(...
            Env, Nwk, Dat, tgt, N, constraints(c,1), ...
            constraints(c,2));
      end%end loop over the possible constraints

      %pack the first three rows of A, store the
      %corresponding parts of b
      A(1,:) = tempA(1,:);
      A(2,:) = tempA(2,:);
      b(1) = tempB(1);
      b(2) = tempB(2);
      A(3,:) = tempA(3,:);%store this vector as the third
         % row of A
      b(3) = tempB(3);%store the entry of b vector
   end%end constraintMatrix
end%end normal public methods

methods(Static)
   function[out] = toaConstraintGraph(Nwk, Dat, T, tgt)
      %validate attributes
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(T, {'numeric'}, ...
         {'nonempty', 'scalar','positive','>=',4,...
         '<=',Nwk.nTowers},...
         '','T')
      validateattributes(tgt,{'numeric'},...
         {'nonempty', 'scalar', 'positive',...
         '<=',Dat.n},'','tgt')
      if Nwk.nTowers < T
         error(['Insufficient towers in network',...
            ' to support analysis.'])
      end

      %Determine the towers will be used for the
      %analysis. Ensure
      power = Dat.power(tgt,:);%get the power recieved at
         % each tower
      temp = sort(power, 'descend');
```

```
         %if the number of towers to consider is less than
         %the total number avaliable
         if T < Nwk.nTowers
            temp = 0.5*(temp(T)+temp(T+1));
         else
            temp = min(temp);
         end
         thresh = min(Nwk.detectThreshold, temp);

         %delete all of the towers which cannot hear the
         %transmitter based on the detection threshold.
         towers = 1:1:Nwk.nTowers;%start with the set of all
            % towers
         for n = Nwk.nTowers:-1:1%go backwards to make this
               %strategy work
            if power(n)<thresh%if insufficient recieved
                  %signal
               towers(n) = [];%delete that tower index
            end%end if
         end%for each tower

         %initialize a Graph object with the remaining
         %towers
         cGraph = Geolocation.Analysis.Graph(towers);
         %compute and store the weights associated with each
         %edge
         tPairs = combnk(towers,2);
         nTP = length(tPairs(:,1));
         for p = 1:1:nTP
            %the weight is 1 so no a priori information is
            %used in selecting constraint equations
            w = 1;
            %store to the graph
            setEdge(cGraph, tPairs(p,1), tPairs(p,2), w)
         end%end loop over pairs

         %Output the result
         out = findTree(cGraph);
      end%end toaConstraintGraph
   end%end static methods
end
```

12. TOA4A Class

```
classdef TOA4A < Geolocation.Analysis.TOA
   %TOA4A Time Of Arrival Position Solver
   %   This class provides an algorithm to solve for the
   %   position of a target in three dimensions given
   %   objects of class Network and Data along with an
   %   estimated refractivity. This class includes the use
   %   of a edge selection process. This class cannot handle
   %   missing observations.
   %
   %Inputs:
   %  Nwk: an object of class Network
   %  Dat: an object of class Data
   %  N: assumed refractivity in parts per million
   %  T: number of towers to consider when forming the final
   %  solution (>=4) This variable forces at least T towers
   %  to be considered in the analysis. If the Nwk's
   %  detection threshold is lower than that required to
   %  keep T towers, then more are used.
   %
   %Outputs:
   %  position: an Mx3 matrix of estimated positions.
   %
   %FUNCTIONS
   %TOA4A(Nwk, Dat, N, T): The constructor. Takes arguments
   %  Nwk: an object of class Geoloction.Network
   %  Dat: an object of class Geoloction.Data
   %  N: the assumed refractivity in parts per million
   %  T: number of towers to consider when forming the final solution
   %
   %constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
   %constraints) returns the set of ``best" (in this class,
   %no weights) constraints packed into matrix form for
   %solving position estimation problem. Takes inputs
   %  obj: an object of class TOA4
   %  Nwk: an object of class Network
   %  Env: an object of class Geolocation.Environment
   %  Dat: an object which inherits from class Data
   %  N: the assumed refractivity in parts per million
   %  tgt: the target number to consider
   %  constraints: a list of tower pairs which are the
   %  possible constraints
```

```
%
%toaConstraintGraph(Nwk, Dat, T, tgt): a static method
%which returns a TOA Constraint Graph which is a list of
%the tower pairs to use in forming the final solution.
%Takes arguments
%  Nwk: an object of class Geoloction.Network
%  Dat: an object of class Geoloction.Data
%  N: the assumed refractivity in parts per million
%  T: number of towers to consider when forming the final
%  solution
%  tgt: the target number to consider
%

properties(SetAccess = immutable)
   position
   N
   T
end

methods
   function[obj] = TOA4A(Nwk, Dat, Env, N, T)
      %validate object inputs
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'}, 'TOA','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'}, 'TOA','Dat')

      %record the refractivity
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      obj.N = N;

      %validate and record T
      validateattributes(T, {'numeric'}, ...
         {'integer', '>=', 4,'<=',Nwk.nTowers},...
         'TOA4', 'T')
      obj.T = T;

      %initialize the position matrix
      obj.position = zeros(Dat.n, 3);

      %FOR each target
   for ind = 1:1:Dat.n
      %Select a set of independant constraints
      constraints = ...
         obj.toaConstraintGraph(Nwk, Dat, T, ind);
      %Select those constraints which maximimze the
      %determinant of the constraint matrix
      [A,b] = constraintMatrix(obj, ...
         Env, Nwk, Dat, N, ind, constraints);

      %Solve the matrix equation and save
      obj.position(ind, :) = A\b;
   end%end FOR each target
end%End constructor

function[A,b] = constraintMatrix(obj, ...
      Env, Nwk, Dat, N, tgt, constraints)
   %validate attributes
   validateattributes(obj, ...
      {'Geolocation.Analysis.TOA4A'},...
      {'nonempty'},...
      '','obj')
   validateattributes(Nwk, {'Geolocation.Network'},...
      {'nonempty'},...
      '','Nwk')
   validateattributes(Env, ...
      {'Geolocation.Environment'},...
      {'nonempty'},...
      '','Env')
   validateattributes(Dat, {'Geolocation.Data'},...
      {'nonempty'},...
      '','Dat')
   validateattributes(N, {'numeric'}, ...
      {'nonempty', 'scalar'},'TOA','N')
   validateattributes(constraints, {'numeric'},...
      {'nonempty','ncols',2,'integer','positive'},...
      '','constraints')
   %Initialize A and B
   A = zeros(3,3);
   b = zeros(3,1);

   %compute the set of rows of A
   tempA = zeros(length(constraints(:,1)),3);
```

```
tempB = zeros(length(constraints(:,1)),1);
for c = 1:1:length(constraints(:,1))
   [tempA(c,:),tempB(c,1)] = ...
      obj.toaConstraintEquation(...
   Env, Nwk, Dat, tgt, N, constraints(c,1),...
   constraints(c,2));
end%end loop over the possible constraints

%pick the two most orthogonal rows
temp = combnk(1:1:length(constraints(:,1)),2);
tempIP = ...
   zeros(nchoosek(length(constraints(:,1)),2),1);
for c = 1:1:length(temp(:,1))
   tempIP(c) = ...
      abs(dot(tempA(temp(c,1),:),...
      tempA(temp(c,2),:)));
end%end loop over the pairwise combinations of
   %constraints
[~,tempIP] = min(tempIP);%which pair has the
   %smallest dot product
tempIP = temp(tempIP,:);%get constraints are most
   %orthogonal

%pack these rows into the first two rows of A,
%store the corresponding parts of b
A(1,:) = tempA(tempIP(1),:);
A(2,:) = tempA(tempIP(2),:);
b(1) = tempB(tempIP(1));
b(2) = tempB(tempIP(2));

%pick the vector which is most orthogonal to these
%first two
tempC = cross(A(1,:), A(2,:));%the vector
   %orthogonal to the first two
tempCP = zeros(length(constraints(1,:)),1);
for c = 1:1:length(constraints(:,1))
   tempCP(c,1) = abs(dot(tempA(c,:),tempC));
end%test each constraint. I choose to retest the
   %two I already have because I know they will be
   %zero
[~,tempCP] = max(tempCP);%figure out which one is
   % most orthogonal
```

```
      A(3,:) = tempA(tempCP,:);%store this vector as the
         %third row of A
      b(3) = tempB(tempCP);%store the entry of b vector
   end%end constraintMatrix
end%end normal public methods

methods(Static)
   function[out] = toaConstraintGraph(Nwk, Dat, T, tgt)
      %validate attributes
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(T, {'numeric'}, ...
         {'nonempty', 'scalar','positive','>=',4},'','T')
      validateattributes(tgt,{'numeric'},...
         {'nonempty', 'scalar', 'positive',...
         '<=',Dat.n},'','tgt')

      %Determine the towers will be used for the
      %analysis. Ensure
      power = Dat.power(tgt,:);%get the power recieved at
         % each tower
      temp = sort(power, 'descend');
      %if the number of towers to consider is less than
      %the total number avaliable
      if T < Nwk.nTowers
         temp = 0.5*(temp(T)+temp(T+1));
      else
         temp = min(temp);
      end
      thresh = min(Nwk.detectThreshold, temp);

      %delete all of the towers which cannot hear the
      %transmitter based on the detection threshold.
      towers = 1:1:Nwk.nTowers;%start with the set of all
         % towers
      for n = Nwk.nTowers:-1:1%go backwards to make this
            %strategy work
         if power(n)<thresh%if insufficient recieved
```

```
                %signal
                towers(n) = [];%delete that tower index
            end%end if
        end%for each tower

        %initialize a Graph object with the remaining
        %towers
        cGraph = Geolocation.Analysis.Graph(towers);
        %compute and store the weights associated with each
        %edge
        tPairs = combnk(towers,2);
        nTP = length(tPairs(:,1));
        for p = 1:1:nTP
            %set the weight equal to one. Any positive
            %constant will do.
            w = 1;
            %store to the graph
            setEdge(cGraph, tPairs(p,1), tPairs(p,2), w)
        end%end loop over pairs

        %Output the result
        out = findTree(cGraph);
    end%end toaConstraintGraph
end%end static methods

end
```

13. TOA4B Class

```
classdef TOA4B < Geolocation.Analysis.TOA
    %TOA4B Time Of Arrival Position Solver
    %   This class provides an algorithm to solve for the
    %   position of a target in three dimensions given
    %   objects of class Network and Data along with an
    %   estimated refractivity. This class includes the use
    %   of a weight function to augment the edge selection
    %   process and the weight is the difference as a
    %   fraction of the total. This class cannot handle
    %   missing observations.
    %
    %Inputs:
```

```
%  Nwk: an object of class Network
%  Dat: an object of class Data
%  N: assumed refractivity in parts per million.
%  T: number of towers to consider when forming the final
%  solution (>=4) This variable forces at least T towers
%  to be considered in the analysis. If the Nwk's
%  detection threshold is lower than that required to
%  keep T towers, then more are used.
%
%Outputs:
%  position: an Mx3 matrix of estimated positions.
%
%FUNCTIONS
%TOA4B(Nwk, Dat, N, T): The constructor. Takes arguments
%  Nwk: an object of class Geoloction.Network
%  Dat: an object of class Geoloction.Data
%  N: the assumed refractivity in parts per million
%  T: number of towers to consider when forming the final
%  solution
%
%constraintMatrix(obj, Env, Nwk, Dat, N, tgt,
%constraints) returns the set of ``best" (in this class,
%no weights) constraints packed into matrix form for
%solving position estimation problem. Takes inputs
%  obj: an object of class TOA4
%  Nwk: an object of class Network
%  Env: an object of class Geolocation.Environment
%  Dat: an object which inherits from class Data
%  N: the assumed refractivity in parts per million
%  tgt: the target number to consider
%  constraints: a list of tower pairs which are the
%  possible constraints
%
%toaConstraintGraph(Nwk, Dat, T, tgt): a static method
%which returns a TOA Constraint Graph which is a list of
%the tower pairs to use in forming the final solution.
%Takes arguments
%  Nwk: an object of class Geoloction.Network
%  Dat: an object of class Geoloction.Data
%  N: the assumed refractivity in parts per million
%  T: number of towers to consider when forming the final
%  solution
%  tgt: the target number to consider
%

properties(SetAccess = immutable)
   position
   N
   T
end

methods
   function[obj] = TOA4B(Nwk, Dat, Env, N, T)
      %validate object inputs
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         'TOA','Nwk')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         'TOA','Dat')

      %record the refractivity
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      obj.N = N;

      %validate and record T
      validateattributes(T, {'numeric'},...
         {'integer', '>=', 4, '<=',Nwk.nTowers},...
         'TOA4', 'T')
      obj.T = T;

      %initialize the position matrix
      obj.position = zeros(Dat.n, 3);

      %FOR each target
      for ind = 1:1:Dat.n
         %Select a set of independant constraints
         constraints = obj.toaConstraintGraph(Nwk, ...
            Dat, Env, T, N, ind);
         %Select those constraints which maximimze the
         %determinant of the constraint matrix
         [A,b] = constraintMatrix(obj, ...
            Env, Nwk, Dat, N, ind, constraints);
         %Solve the matrix equation and save
         obj.position(ind, :) = A\b;
      end%end FOR each target
   end%End constructor

   function[A,b] = constraintMatrix(obj, ...
         Env, Nwk, Dat, N, tgt, constraints)
      %validate attributes
      validateattributes(obj, ...
         {'Geolocation.Analysis.TOA4B'},...
         {'nonempty'},...
         '','obj')
      validateattributes(Nwk, {'Geolocation.Network'},...
         {'nonempty'},...
         '','Nwk')
      validateattributes(Env, ...
         {'Geolocation.Environment'},...
         {'nonempty'},...
         '','Env')
      validateattributes(Dat, {'Geolocation.Data'},...
         {'nonempty'},...
         '','Dat')
      validateattributes(N, {'numeric'}, ...
         {'nonempty', 'scalar'},'TOA','N')
      validateattributes(constraints, {'numeric'},...
         {'nonempty','ncols',2,'integer','positive'},...
         '','constraints')
      %Initialize A and B
      A = zeros(3,3);
      b = zeros(3,1);

      %compute the set of rows of A
      tempA = zeros(length(constraints(:,1)),3);
      tempB = zeros(length(constraints(:,1)),1);
      for c = 1:1:length(constraints(:,1))
         [tempA(c,:),tempB(c,1)] = ...
            obj.toaConstraintEquation(...
         Env, Nwk, Dat, tgt, N, constraints(c,1), ...
         constraints(c,2));
      end%end loop over the possible constraints
```

```
      %pick the two most orthogonal rows
      temp = combnk(1:1:length(constraints(:,1)),2);
      tempIP = ...
         zeros(nchoosek(length(constraints(:,1)),2),1);
      for c = 1:1:length(temp(:,1))
         tempIP(c) = ...
            abs(dot(tempA(temp(c,1),:),...
            tempA(temp(c,2),:)));
      end%end loop over the pairwise combinations of
         %constraints
      [~,tempIP] = min(tempIP);%which pair has the
         %smallest dot product
      tempIP = temp(tempIP,:);%get constraints are most
         %orthogonal

      %pack these rows into the first two rows of A,
      %store the corresponding parts of b
      A(1,:) = tempA(tempIP(1),:);
      A(2,:) = tempA(tempIP(2),:);
      b(1) = tempB(tempIP(1));
      b(2) = tempB(tempIP(2));

      %pick the vector which is most orthogonal to these
      %first two
      tempC = cross(A(1,:), A(2,:));%the vector
         %orthogonal to the first two
      tempCP = zeros(length(constraints(1,:)),1);
      for c = 1:1:length(constraints(:,1))
         tempCP(c,1) = abs(dot(tempA(c,:),tempC));
      end%test each constraint. I choose to retest the
         %two I already have because I know they will be
         %zero
      [~,tempCP] = max(tempCP);%figure out which one is
         % most orthogonal
      A(3,:) = tempA(tempCP,:);%store this vector as the
         %third row of A
      b(3) = tempB(tempCP);%store the entry of b vector
   end%end constraintMatrix
end%end normal public methods

methods(Static)
   function[out] = toaConstraintGraph(Nwk, Dat, Env,...
   T, N, tgt)
%validate attributes
validateattributes(Nwk, {'Geolocation.Network'},...
   {'nonempty'},...
   '','Nwk')
validateattributes(Dat, {'Geolocation.Data'},...
   {'nonempty'},...
   '','Dat')
validateattributes(T, {'numeric'}, ...
   {'nonempty', 'scalar','positive','>=',4,...
   '<=',Nwk.nTowers},...
   '','T')
validateattributes(N, {'numeric'}, ...
   {'nonempty', 'scalar'},'TOA','N')
validateattributes(tgt,{'numeric'},...
   {'nonempty', 'scalar', 'positive',...
   '<=',Dat.n},'','tgt')

%Determine the towers will be used for the
%analysis. Ensure
power = Dat.power(tgt,:);%get the power recieved at
   % each tower
temp = sort(power, 'descend');
%if the number of towers to consider is less than
%the total number avaliable
if T < Nwk.nTowers
   temp = 0.5*(temp(T)+temp(T+1));
else
   temp = min(temp);
end
thresh = min(Nwk.detectThreshold, temp);

%delete all of the towers which cannot hear the
%transmitter based on the detection threshold.
towers = 1:1:Nwk.nTowers;%start with the set of all
   % towers
for n = Nwk.nTowers:-1:1%go backwards to make this
      % strategy work
   if power(n)<thresh%if insufficient recieved
         % signal
      towers(n) = [];%delete that tower index
   end%end if
```

```
         end%for each tower

         %initialize a Graph object with the remaining
         %towers
         cGraph = Geolocation.Analysis.Graph(towers);
         %compute and store the weights associated with
         % each edge
         tPairs = combnk(towers,2);
         nTP = length(tPairs(:,1));
         for p = 1:1:nTP
            %compute the fractional distance from tower 1 to
            %tower 2
            temp1 = Dat.timingAdjust(tgt,tPairs(p,1));
            temp2 = Dat.timingAdjust(tgt,tPairs(p,2));
            vp = Env.c/Env.ppm2n(N);
            ri = vp*temp1*Nwk.tau;
            rj = vp*temp2*Nwk.tau;
            vij = Nwk.towers(tPairs(p,1),:) - ...
               Nwk.towers(tPairs(p,2),:);
            dij = sqrt(dot(vij,vij));
            tempTop = ri^2 - rj^2 + dot(vij,vij);
            tempBot = 2*dij;
            di = tempTop/tempBot;
            temp4 = di./dij;
            %the total weight is the difference as a
            %fraction of the total.
            w = max(temp4, 1-temp4);
            %store to the graph
            setEdge(cGraph, tPairs(p,1), tPairs(p,2), w)
         end%end loop over pairs

         %Output the result
         out = findTree(cGraph);
      end%end toaConstraintGraph
   end%end static methods
end
```

14. VelocityError Class

```
classdef VelocityError
   %VELOCITYERROR Provides methods to analyze errors in
```

```
%estimated target velocities
%   VelocityError provides methods to analyze the errors
%   associated with estimated velocity estimates given
%   the true values.

properties (SetAccess = private)
   error;%error
   L2;%L2 norm of the error vectors
   L1;%L1 norm of the error vectors
   Linf;%L0 norm of the error vectors
end%end properties

methods
   % Constructor
   %FrequencyError--creates and initializes an object of
   %class PositionError
   %Tgt: An object of Class Target containing the true
   %data
   %Est: Any object which contains a property position
   %which contains the estimated positions and a property
   %N the assumed refractivity of the environment
   function[obj] = VelocityError(Tgt, Est)
      obj.error = Tgt.velocity - Est.velocity;
      obj.L2 = ...
         sqrt(diag(obj.error*transpose(obj.error)));
      obj.L1 =  ...
         transpose(sum(transpose(abs(obj.error))));
      obj.Linf =  ...
         transpose(max(transpose(abs(obj.error))));
   end%end PositionError

   % Plot Methods
   %l2histogram--plots the histogram of the L2 errors
   %obj: an object of class PositionError
   %error: a string, the name of the error measure
   %property to be plotted
   %Est: any object of class which contains the assumed
   %refractivity
   %Env: an object of class Environment
   %bins: number of bins to use or a vector of bins
   %newFig: boolean to indicate whether or not to write
   %to a new figure
```

```
      function[] = l2histogram(obj, error, Est, Env,...
            bins, newFig)
         %switch on error to pick the correct set of errors
         switch error
            case 'L2' %L2 errors
               err = obj.L2;
               title = 'Frequency Adjust';
               pick = 'L2';
            case 'L1' %L1 errors
               err = obj.L1;
               title = 'Frequency Adjust';
               pick = 'L1';
            case 'Linf' %L-infinity errors
               err = obj.Linf;
               title = 'Frequency Adjust';
               pick = 'L-infinity';
         end
         %compute the proportions in each bin
         [n, xout] = hist(err, bins);
         n = n/sum(n);

         %start a new figure as appropriate
         if newFig == true
            figure()
         end

         %create the histogram
         bar(xout, n, 1)
         xlabel('Error (Meters)')
         ylabel('Proportion')
         title({['Histogram of ',title,...
            ' Velocity Esimate ', pick, ' Errors'];
            ['True Refractivity: ',...
            num2str(round(getRefractivity(Env)))];
            ['Assumed Refractivity: ',...
            num2str(round(Est.N))]});
      end%end l2histogram
   end%end methods
end
```

F. UTILITY FUNCTIONS

1. parPresH20.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%parPresH2O.m -- Partial Pressure of Water in the atmosphere
%
%J. Q. McClintic, 2012
%
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius
%   dewpoint: current dewpoint temperature in celcius
%
%Outputs: PPW: partial pressure of water in millibars
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = parPresH2O(ATP, temp, dewpoint)
    % Temperature Check
    % If the temp is greater than 0 deg C, use the water
    % curve, else the ice curve
    if temp > 0
        PPW = waterCurve(ATP, dewpoint);
    else
        PPW = iceCurve(ATP, dewpoint);
    end %end if
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%waterCurve.m -- A function to compute the partial pressure
%  of water assuming the air temperature is greater than 0
%  degrees C
%
%Buck, A. New Equations for Computing Vapor Pressure and
%Enchancement Factor. Journal Of Applied Meteorology.
%December 1981, 1527-32.
%
%J. Q. McClintic, 2012
%
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius. ambient for pure water
%     vapor, else dewpoint temperature
```

```
%
%Outputs: PPW: partial pressure of water in millibars
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = waterCurve(ATP, temp)
    % Compute the unenchanced partial pressure of water

    %declare the various parameters table 2, curve ew6
    a = 6.1121;
    b = 18.564;
    c = 255.57;
    d = 254.4;

    % compute the partial pressure
    Ew = a*exp( temp*(b - temp/d)/(temp + c) );

    % Compute the enhancement factor

    % declare the various parameters
    A = 7.2e-4;
    B = 3.2e-6;
    C = 5.9e-10;
    D = 0; %included in case the choice of
    E = 0; % cuves were to change

    % compute the enhancement factor
    f = 1 + A + ATP*(B + C*(temp + D + E*ATP)^2);

    % Compute the full partial pressure
    PPW = Ew*f;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%iceCurve.m -- A function to compute the partial pressure of
%   water assuming the air temperature is less than 0
%   degrees C
%
%Buck, A. New Equations for Computing Vapor Pressure and
%Enchancement Factor. Journal Of Applied Meteorology.
%December 1981, 1527-32.
%
%J. Q. McClintic, 2012
%
```

```
%Inputs:
%   ATP: atmospheric pressure in millibars
%   temp: temperature in celcius. ambient for pure water
%     vapor, else dewpoint temperature
%
%Outputs: PPW: partial pressure of water in millibars
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[PPW] = iceCurve(ATP, temp)
    % Compute the unenchanced partial pressure of water

    %declare the various parameters table 2, curve ei2
    a = 6.1115;
    b = 22.452;
    c = 272.55;

    % compute the partial pressure,
    Ei = a*exp( b*temp/(temp + c) );

    % Compute the enhancement factor

    % declare the various parameters
    A = 3e-4;
    B = 4.18e-6;
    C = 0; %included in case the choice of
    D = 0; % cuves were to change
    E = 0;

    % compute the enhancement factor
    f = 1 + A + ATP*(B + C*(temp + D + E*ATP)^2);

    % Compute the full partial pressure
    PPW = Ei*f;
end
```

2. refract.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% refract.m -- Refractivity Estimate based on Local
% Environmental Conditions
%
```

```
% Author: LTJG J. Q. McClintic, 6FEB12
%
% Inputs:
%   temp--ambient temperature in degrees celcius
%   dewpoint--dewpoint temperature in degrees celcius
%   pressure--total atmospheric pressure in millibar
%
% Output: N--refractivity in parts per million assuming 375
% ppm atmospheric carbon dioxide content
%
% Note: uses Rueger (2002) as cited in the thesis as the
% basis for the formula
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[N] = refract(temp, dewpoint, pressure)

% convert the total pressure to dry air and water vapor
% pressure
Pw = parPresH2O(pressure, temp, dewpoint); %partial pressure
% of water
Pd = pressure - Pw;%dry air is what is left

% Compute the refractivity estimate
K1 = 77.6890;%dry air coeffient
K2 = 71.2952;%wet air linear coefficient
K3 = 375463;%wet air second term coefficient
DK = temp+273.15;%convert temperature to degrees kelvin
N = K1*Pd/DK + K2*Pw/DK + K3*Pw/(DK^2);

end %End refract.m
```

3. refractiveProfile.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% refractiveProfile.m -- Refractivity Profile based on
% environmental inputs
%
% LTJG J. Q. McClintic, 6FEB12
%
% Inputs:
%   temp--a vector of temperatures to plot over (100>=
%   degrees C >=-50)
%   dewpoint--a vector of dewpoint spreads to plot over (>=0
%   degrees C)
%   pressure--total atmospheric pressures (millibars)
%   levels--number of levels to plot or a vector of levels
%   to plot
% Outputs: a plot of refractivity curves, one for each
% pressure specified
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[] = refractiveProfile(temp, dewpoint,...
   pressure, levels)

% declare variables
nVals = zeros(length(temp), length(dewpoint));
%somwhere to hold all the data

% loop over each variable to make the proper curves
for t = 1:1:length(temp)%each tempurature specified
   for d = 1:1:length(dewpoint)%each dewpoint spread
      %specified
         %compute
         nVals(t,d) = ...
            refract(temp(t), temp(t)-dewpoint(d), pressure);
   end%end of dewpoint loop
end %end of temp loop

%display the output
figure()
[C, h] = contour(temp, dewpoint, transpose(nVals), levels);
title({['Refractivity as a Function of Temperature',...
   ' and Dewpoint Spread'];...
   ['Total Pressure ',num2str(pressure),' millibars']})
xlabel('Temperature (degrees C)')
ylabel('Dewpoint Spread (degrees C)')
set(h,'ShowText','on','TextStep',get(h,'LevelStep'))
end
```

4. velocityDifference.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%velocityDifference.m
```

```
%
%Computes the velocity difference between the speed of light
%in vacuum and in air.
%
%Inputs:
%   nLower: lower bound on refractivity to plot (ppm)
%   nUpper: upper bound on refractivity to plot (ppm)
%   c: the speed of light in air. (meters/second)
%   increment: increment between profile points. Smaller =
%   higher resolution (ppm)
%
%Outputs: a vector with the profile
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[out] = velocityDifference(c, nLower, ...
   nUpper, increment)
    %make output variable
    out = [];
    %make the array index variable
    j = 1;
    %convert refractivity to refractive index
    lower = ppm2n(nLower);
    upper = ppm2n(nUpper);
    delta = increment/10^6;
    for i = lower:delta:upper
        out(j) = c*(1-1/i);
        j = j+1;
    end
end
```

5. **rangeBias.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rangeBias--compute range estimage bias due to refractivity
%
% input:
%  N0: the reference value of refractivity in parts per
%  million
%  N: the assumed value of refractivity in parts per million
%  c: reference speed of light (meters/second)
%  t: the time of flight of the signal (seconds)
%
```

```
% output: bias value in meters
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[bias] = rangeBias(N0, N, c, t)
    n0 = ppm2n(N0);%convert to refractive index
    n = ppm2n(N);%convert to refractive index

    %compute bias
    bias = c*t*(n-n0)/(n*n0);
end
```

6. **rangeBiasProfile.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%rangeBiasProfile.m
%
%plots the velocity difference between the speed of light in
%vacuum and
%  in air.
%
%Inputs:
%   N0: the reference value of refractivity in parts per
%   million
%   nLower: lower bound on refractivity to plot (ppm)
%   nUpper: upper bound on refractivity to plot (ppm)
%   c: vector of the speed of light in air. (each in m/s)
%   t: the time of flight of the signal (seconds) increment:
%   increment between profile points. Smaller = higher
%   resolution (ppm)
%   figNum: figure number for the plot
%
%Outputs: a graph
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[] = rangeBiasProfile(N0, nLower, nUpper, c, t,...
   increment, figNum)
   %Variables to hold the various makers and colors for the
   %different profiles
   colors = ['r', 'g', 'b', 'c', 'm', 'k'];
   markers = ['+', 'o', '*', '.', 'x'];

   %generate profile x positions
   x = nLower:increment:nUpper;

   %make an index counter and an empty vector to take output
   y = zeros(length(x), length(c));

   %generate the bias values
   for k = 1:1:length(c)
     for i = 1:1:length(x)
         y(i, k) = rangeBias(N0, x(i), c(k), t);
     end
   end
   %make the plot
   figure(figNum)
   xlim([nLower, nUpper]);%set the x limits
   ylim([min(min(y)),max(max(y))]);%set the ylimits
   for n = 1:1:length(c)
       line(x, y(:,n), 'DisplayName', ...
          ['Speed of Light = ',int2str(c(n)), ' m/s'],...
          'Color', colors(mod(n,6)+1),...
          'Marker', markers(mod(n,5)+1))
   end
   title({'Range Estimate Bias Due to Refraction in Air';...
      ['With a Propogation Time of ',int2str(t*10^6),...
      ' microseconds']})
   xlabel('Refractivity (ppm)')
   ylabel('Range Bias (meters)')
   legend show
   legend('Location','SouthOutside')
end
```

7. rangeBiasTime.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rangeBiasTime--compute range estimage bias as a function
% of time
%  due to a given refractivity
%
% input:
%  N0: the reference value of refractivity in parts per
%  million
%  N: the assumed value of refractivity in parts per million
%  c: reference speed of light (meters/second)
%  tMin: the minimum time of flight of the signal (seconds)
%  tMax: the maximum time of flight of the signal (seconds)
%  tInc: time increment over which to compute
%
% output: bias value in meters and time values in seconds
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[bias, time] = rangeBiasTime(N0, N, c, tMin,...
   tMax, tInc)
    %make the values of time for which to evaluate the bias
    time = tMin:tInc:tMax;
    %make the bias values
    bias = rangeBias(N0, N, c, time);
end
```

8. rangeBiasTimeProfile.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rangeBiasTimeProfile--plot the range bias as a function of
% flight time given a set of refractivity values
%
% input:
%  N0: the reference value of refractivity in parts per
%  million
%  N: a vector of assumed value of refractivity in parts per
%  million
%  c: reference speed of light (meters/second)
%  tMin: the minimum time of flight of the signal
%  (microseconds)
%  tMax: the maximum time of flight of the signal
%  (microseconds)
%  tInc: time increment over which to compute (microseconds)
%  title: whether or not to display the title. Boolean.
%  figNum: the figure number for the plot
%
% output: a plot bias value in meters and time values in
% microseconds
%  for each
%  value of N. Up to thirty different lines are supported
%  via unique color/maker combinations
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[] = rangeBiasTimeProfile(N0, N, c, tMin, tMax,...
   tInc, title,figNum)
validateattributes(N0, {'numeric'}, ...
   {'nonempty', 'scalar'},'','N0')
validateattributes(N, {'numeric'}, ...
   {'nonempty', 'vector'},'','N')
validateattributes(c, {'numeric'}, ...
   {'nonempty', 'scalar', 'positive'},'','c')
validateattributes(tMin, {'numeric'}, ...
   {'nonempty', 'scalar', 'positive'},'','tMin')
validateattributes(tMax, {'numeric'}, ...
   {'nonempty', 'scalar', 'positive','>',tMin},'','tMax')
validateattributes(tMax, {'numeric'}, ...
   {'nonempty', 'scalar', 'positive'},'','tInc')
validateattributes(title,{'logical'},...
   {'nonempty'},'','title')
validateattributes(N0, {'numeric'}, ...
   {'nonempty', 'scalar'},'','figNum')

bias = zeros(length(N), length(tMin:tInc:tMax));
time = tMin:tInc:tMax;
colors = ['r', 'g', 'b', 'c', 'm', 'k'];
markers = ['+', 'o', '*', '.', 'x'];

for n = 1:1:length(N)
   [bias(n, :),~] = ...
      rangeBiasTime(N0, N(n), c, ...
      tMin/10^6, tMax/10^6, tInc/10^6);
end

figure(figNum)
xlim([tMin, tMax])
ylim([min(min(bias)), max(max(bias))])
%make the lines for each curve
for n = 1:1:length(N)
   line(time, bias(n,:), 'DisplayName', ...
      ['N = ',int2str(N(n))], ...
      'Color', colors(mod(n,6)+1), ...
      'Marker', markers(mod(n,5)+1))
end
```

```
if title
   title({['Range Bias Against Propagation',...
      ' Time as a Function of Refractivity'];...
      ['Speed of Light = ',int2str(c),' meters/second']})
end
xlabel('Propogation Time(microseconds)')
ylabel('Range Bias (meters)')
legend show
legend('Location','EastOutside')
end
```

9. environmentalRangeBiasProfile.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% environmentalRangeBiasProfile.m --
%  Refractivity Profile based on environmental inputs
%
% LTJG J. Q. McClintic, 6FEB12
%
% Inputs:
%   temp--a vector of temperatures to plot over (100>=
%   degrees C >=-50)
%   dewpoint--a vector of dewpoint spreads to plot over (>=0
%   degrees C)
%   pressure--total atmospheric pressures (millibars)
%   levels--number of levels to plot or a vector of levels
%   to plot
%  assumedN--assumed level of refractivity locally (ppm)
%  time--reference flight time for signal (microseconds)
%  SOL--assumed speed of light in vacuum (meters/second)
% Outputs: a plot of range bias curves
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[] = environmentalRangeBiasProfile(temp, ...
   dewpoint, pressure, levels, assumedN, time, SOL)

% declare variables
bias = zeros(length(temp), length(dewpoint));
%somwhere to hold all the data

% loop over each variable to make the proper curves
```

```
for t = 1:1:length(temp)%each tempurature specified
   for d = 1:1:length(dewpoint)%each dewpoint spred
      %specified compute
         N = ...
            refract(temp(t), temp(t)-dewpoint(d), pressure);
         bias(t,d) = rangeBias(assumedN, N, SOL, time/10^6);
   end%end of dewpoint loop
end %end of temp loop

% display the output
figure()
[~, h] = contour(temp, dewpoint, transpose(bias), levels);
title({['Range Bias (meters) as a Function of',...
   ' Temperature and Dewpoint Spread'];...
   ['Total Pressure ',num2str(pressure),' millibars'];
   ['Nominal Flight Time: ', num2str(time),...
   ' microseconds'];['Reference Refractivity: ', ...
   num2mstr(assumedN), ' ppm'];
   ['Speed of Light: ', num2str(SOL), ' m/s']})
xlabel('Temperature (degrees C)')
ylabel('Dewpoint Spread (degrees C)')
set(h,'ShowText','on','TextStep',get(h,'LevelStep'))
end
```

10. rb2tauPlot.m

```
function [  ] = rb2tauPlot( tauRange, N0, NHatRange , ...
resolution, contours, refBW)
%rb2tauPlot Plots the ratio of range bias and timing advance
%unit error standard deviation. The ratio is given by
%10e-6*sqrt(12)*(n - n0)/(n*tau) where n is the assumed
%refractive index, n0 is the true refractive index, and tau
%is the timing adjust unit in seconds.
%   Inputs:
%     tauRange--a 2x1 vector of the min and max timing
%     advance unit
%     N0--The reference refractivity
%     NHatRange--a 2x1 vector of the min and max
%     refractivity to plot resolution--the number of points
%     to plot in each direction contours--either the number
%     of contours to plot or a vector of levels. Is passed
%     directly to contour.m.
%     refBW--a possibly empty vector of system bandwidths to
%     plot as references in MHz.

%validate attributes
validateattributes(tauRange,{'numeric'},...
   {'nonempty','vector','numel',2,'positive'},'','tauRange')
validateattributes(N0,{'numeric'},...
   {'nonempty','scalar'},'','N0')
validateattributes(NHatRange,{'numeric'},...
   {'nonempty','vector','numel',2},'','tauRange')
validateattributes(resolution,{'numeric'},...
   {'nonempty','scalar','positive'},'','resolution')
validateattributes(contours,{'numeric'},...
   {'nonempty'},'','contours')
if nargin == 6%that is, if refBW is provided
   validateattributes(refBW,{'numeric'},...
      {'vector','positive'},'','refBW')
end%

%force tauRange and NHatRange to be properly ordered
tauRange = sort(tauRange);
NHatRange = sort(NHatRange);

%compute the lower and upper limits of refractive index and
%the reference refractive index
N0 = 1+N0/1e6;
NHatRange = 1+NHatRange/1e6;

%compute the values for the axes of the plot
yVals = linspace(NHatRange(1),NHatRange(2),resolution);
xVals = linspace(tauRange(1),tauRange(2),resolution);

%compute the matrix of values
rMat = zeros(length(xVals),length(yVals));
for x = 1:1:length(xVals)
   for y = 1:1:length(yVals)
      %The x and y need to be flipped because contour uses
      rMat(y,x) = ...
         (sqrt(12)*10e-6)*(yVals(y) - N0)/...
         (xVals(x)*yVals(y));
   end%loop over y values
```

```
end%loop over x values

%Generate the contour plot
[~,h] = contour(xVals.*1e9, (yVals-1).*1e6,...
   rMat, contours,...
      'DisplayName','b_d / \sigma\{\epsilon_\tau\}');
set(h,'ShowText','on','TextStep',get(h,'LevelStep')*2)
ylabel('Refractivity (ppm)')
xlabel('Timing Advance Unit (nanoseconds)')

%add the refBW lines
if nargin == 6
   colors = ['r', 'g', 'b', 'c', 'm', 'k'];
   markers = ['+', 'o', '*', '.', 'x'];
   hold on;
   for ind = 1:1:length(refBW)
      %determine the equivilent timing adjust unit
      tau = timingAdjustUnit(refBW(ind));
      %plot a vertical line with x = timing adjust unit from
      %the bottom of the refractivity range to the top of
      %the refractivity range
      line([tau tau].*1e9, ...
         (NHatRange - 1).*10^6,...
         'DisplayName', ...
         ['BW = ',int2str(refBW(ind)),'MHz'], ...
         'Color', colors(mod(ind,6)+1), 'Marker',...
         markers(mod(ind,5)+1));
   end%end loop over refBW values
end%add the refBW lines

%show the legend
legend show 'Location' Best

end%rb2tauPlot

function[n] = sampFactor(BW)
if mod(BW, 1.75)==0
   n = 8/7;
elseif mod(BW, 1.25)==0
   n = 28/25;
else
   n = 8/7;
```

```
end%end if
end%sampFactor

function[tau] = timingAdjustUnit(BW)
   tau = 1/(floor(1e6*sampFactor(BW)*BW/8000)*8000);
end%timingAdjustUnit
```

C.   EXAMPLE SIMULATION SCRIPT

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation Script
%
% J. Q. McClintic
%
% 01 DEC 12
%
% NOTE: THIS SCRIPT WILL CLEAR YOUR MATLAB WORKSPACE AND
% TERMINAL
%
% INSTRUCTIONS: Simulation parameters are found in the
% appropriate cell with instructions. Any parameters common
% to multiple cells are in the ``Simulation Parameters" cell
%
% INPUTS: See the comments associated with the various
% settable parameters (parameter names are in ALL_CAPS
% unless otherwise specified)
%
% OUTPUTS: (some may be commented out and ergo not
% collected)
% meanL2: a vector of the mean L2 error for each run
% meanL1: a vector of the mean L1 error for each run
% meanLinf: a vector of the mean L-infinity error for each
% run
% sdL2: a vector of the standard deviation of the L2 error
% for each run
% sdL2: a vector of the standard deviation of the L1 error
% for each run
% sdL2: a vector of the standard deviation of the L
% -infinity error for each run
% medianL2: a vector of the median L2 error for each run
% medianL1: a vector of the median L1 error for each run
% medianLinf: a vector of the median Linf error for each run
% minL2: a vector of the minimum L2 error for each run
% minL1: a vector of the minimum L1 error for each run
% minLinf: a vector of the minimum L-infinity error for each
% run
% maxL2: a vector of the minimum L2 error for each run
% maxL1: a vector of the minimum L1 error for each run
% maxLinf: a vector of the minimum L-infinity error for each
% run
% skewL2: a vector of the skew of the L2 errors for each run
% skewL1: a vector of the skew of the L1 errors for each run
% skewLinf: a vector of the skew of the L-infinity errors
% for each run
% kurtL2: a vector of the kurtosis of the L2 errors for each
% run
% kurtL1: a vector of the kurtosis of the L1 errors for each
% run
% kurtLinf: a vector of the kurtosis of the L-infinity
% errors for each run
% iqrL2: interquartile range of L2 errors
% iqrL1: interquartile range of L1 errors
% iqrLinf: interquartile range of Linf errors
%
% NOTE: The workspace is saved at the end of each completed
% run after deleting the excess objects. This means that in
% the event of failure some data still remains. The path to
% the file to be saved to must be set manually at the end of
% this file.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc

%% Simulation Parameters
N_RUNS = 1000; % Number of runs of the basic simulation to
%be averaged
TYPE = 'TOA4'; % Sets whether to look at 'POSITION' or
%'VELOCITY' error
rng(2)%set the initial seed to a fixed, arbitrary value
seeds = randi(N_RUNS, [N_RUNS,1]);%generate the seed for
%each run

%% Set up the Environment
%Parameters
AIR_PRESS = 1000;% Air pressure in millibar
TEMP = 30; %Ambient air pressure in degrees Celcius
DEWPOINT = 20; %Dewpoint temperature in degrees Celcius

%Set up the environment object
```

```
ENV = Geolocation.Environment(AIR_PRESS, TEMP, DEWPOINT);

%% Refractivity Settings
% Uncomment one of the following lines
N = getRefractivity(ENV); % actual refractivity of the
   %simulated atmosphere
%N = -692; % assume speed of light is 3e8 m/s
%N = 0; % assume NIST true speed of light

%% Set up the Generic Network
% Network Parameters
N_TOWERS = 50; % Number of towers
BANDWIDTH = 10; % Bandwidth of any tower in Megahertz
N_USED = 1024; % Number of subcarriers used (including DC
   %subcarrier)
G = 1/32; % Ratio of CP time to ``useful" time.

% Set up the network
NWK = Geolocation.Network(N_TOWERS, BANDWIDTH, N_USED, G);

%% Tower locations
% If random tower placement is desired for each run,
% uncomment this line and comment out
% the other commands in this cell

RANDOM_TOWERS = true;
MAX_X = 10000;%Max +/- value of the x-component of position
MAX_Y = 10000;%Max +/- value of the y-component of position
MAX_Z = 100;%Max +/- value of the z-component of position

% If you want to place towers is specific locations, then
% comment out the commands above in this cell, uncomment the
% ones below, and provide a placeTower command call for each
% tower you want. Locations are given in cartesian
% corrdinates in the second argument as a vector.

% placeTower(NWK, [5000 5000 100], 1);
% placeTower(NWK, [-5000 -5000 -100], 2);
% placeTower(NWK, [5000 -5000 50], 3);
% placeTower(NWK, [-5000 5000 -50], 4);
% placeTower(NWK, [0 0 0], 5);
```

```
%% Target Parameters
% Set the target parameters that will be use for each trial
% run.
N_TARGETS = 5000; % Number of targets to create
TARGET_LIMS = [10000, 10000, 100]; % +/- x, y, and z limits
   %on position
VELOCITY = 10; % max velocity in any direction
CENTER_F = [5.725e9 5.875e9]; % target center transmitted
   %frequency

%% Simulations Loop
% Set up places to hold the various statistics
meanL2 = zeros(N_RUNS, 1);
meanL1 = zeros(N_RUNS, 1);
meanLinf = zeros(N_RUNS, 1);
sdL2 = zeros(N_RUNS, 1);
sdL1 = zeros(N_RUNS, 1);
sdLinf = zeros(N_RUNS, 1);
medianL2 = zeros(N_RUNS, 1);
medianL1 = zeros(N_RUNS, 1);
medianLinf = zeros(N_RUNS, 1);
minL2 = zeros(N_RUNS, 1);
minL1 = zeros(N_RUNS, 1);
minLinf = zeros(N_RUNS, 1);
maxL2 = zeros(N_RUNS, 1);
maxL1 = zeros(N_RUNS, 1);
maxLinf = zeros(N_RUNS, 1);
skewL2 = zeros(N_RUNS, 1);
skewL1 = zeros(N_RUNS, 1);
skewLinf = zeros(N_RUNS, 1);
kurtL2 = zeros(N_RUNS, 1);
kurtL1 = zeros(N_RUNS, 1);
kurtLinf = zeros(N_RUNS, 1);
iqrL2 = zeros(N_RUNS, 1);
iqrL1 = zeros(N_RUNS, 1);
iqrLinf = zeros(N_RUNS, 1);

% Loop through all the simulation runs

h = waitbar(0,'Please wait...');% set up a waitbar
for run = 1:1:N_RUNS
   %waitbar
```

```
waitbar(run/N_RUNS)

%seed the random number generator for this run
rng(seeds(run))

% make random towers for the run if desired by user
if RANDOM_TOWERS == true
   randomTowers(NWK, [MAX_X, MAX_Y, MAX_Z]);
end

% make the targets for this run
TGT = Geolocation.Target(N_TARGETS);
randomTarget(TGT, TARGET_LIMS, VELOCITY, CENTER_F)

% make the dataset
DATA = Geolocation.Data(ENV, NWK, TGT);

% compute estimates and errors for desired estimate type
switch TYPE
   case 'TOA4'
      ESTIMATE = ...
         Geolocation.Analysis.TOA4(ENV, NWK, DATA, N);
      ERROR = ...
         Geolocation.Analysis.PositionError(TGT,...
         ESTIMATE);
   case 'TOA4A'
      ESTIMATE = ...
         Geolocation.Analysis.TOA4A(ENV, NWK, DATA, N);
      ERROR = ...
         Geolocation.Analysis.PositionError(TGT,...
         ESTIMATE);
   case 'TOA4B'
      ESTIMATE = ...
         Geolocation.Analysis.TOA4B(ENV, NWK, DATA, N);
      ERROR = ...
         Geolocation.Analysis.PositionError(TGT,...
         ESTIMATE);
   case 'TDOA4'
      ESTIMATE = ...
         Geolocation.Analysis.TDOA5(ENV, NWK, DATA, N);
      ERROR = ...
         Geolocation.Analysis.PositionError(TGT, ...
            ESTIMATE);
    case 'TDOA4A'
        ESTIMATE = ...
            Geolocation.Analysis.TDOA5A(ENV, NWK, DATA, N);
        ERROR = ...
            Geolocation.Analysis.PositionError(TGT, ...
            ESTIMATE);
    case 'TDOA4B'
        ESTIMATE = ...
            Geolocation.Analysis.TDOA5B(ENV, NWK, DATA, N);
        ERROR = Geolocation.Analysis.PositionError(TGT, ...
            ESTIMATE);
    case 'DOPPLER4'
        ESTIMATE = ...
            Geolocation.Analysis.Doppler4(ENV, NWK,...
            DATA, N);
        ERROR = Geolocation.Analysis.VelocityError(TGT, ...
            ESTIMATE);
    case 'DOPPLER4A'
        ESTIMATE = ...
            Geolocation.Analysis.Doppler4A(ENV, NWK,...
            DATA, N);
        ERROR = Geolocation.Analysis.VelocityError(TGT, ...
            ESTIMATE);
    case 'DOPPLER4B'
        ESTIMATE = ...
            Geolocation.Analysis.Doppler4B(ENV, NWK,...
            DATA, N);
        ERROR = Geolocation.Analysis.VelocityError(TGT,...
            ESTIMATE);
    otherwise
        error('Invalid Analysis Type.')
end% end switch for analysis type

% compute and place the various statistics in their
% holding vectors
meanL2(run) = mean(ERROR.L2);
%meanL1(run) = mean(ERROR.L1);
%meanLinf(run) = mean(ERROR.Linf);
sdL2(run) = std(ERROR.L2);
%sdL1(run) = std(ERROR.L1);
%sdLinf(run) = std(ERROR.Linf);
```

```
    medianL2(run) = median(ERROR.L2);
   %medianL1(run) = median(ERROR.L1);
   %medianLinf(run) = median(ERROR.Linf);
    %minL2(run) = min(ERROR.L2);
    %minL1(run) = min(ERROR.L1);
    %minLinf(run) = min(ERROR.Linf);
    %maxL2(run) = max(ERROR.L2);
    %maxL1(run) = max(ERROR.L1);
    %maxLinf(run) = max(ERROR.Linf);
    %skewL2(run) = skewness(ERROR.L2);
    %skewL1(run) = skewness(ERROR.L1);
    %skewLinf(run) = skewness(ERROR.Linf);
    %kurtL2(run) = kurtosis(ERROR.L2) - 3;
    %kurtL1(run) = kurtosis(ERROR.L1) - 3;
    %kurtLinf(run) = kurtosis(ERROR.Linf) - 3;
    iqrL2(run) = iqr(ERROR.L2);
    %iqrL1(run) = iqr(ERROR.L1);
    %iqrLinf(run) = iqr(ERROR.Linf);

    % clear all the objects no longer needed from this run
    clear TGT
    clear ESTIMATE
    clear ERROR

    %output some status markers
    disp(['Run: ',num2str(run),' of ',...
       num2str(N_RUNS),' complete.'])
    time = clock;
    time(4:6)

    %save the results so far. This will overwrite whatever
    %file with the given name exists in the folder in the
    %path.
    save(['\\special\jqmcclin$\ThesisCode',...
       '\ThesisSims\TOA\test.mat']);
end%end main simulation loop
close(h)% close the waitbar
time = clock;
time(4:6)
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] *IEEE Standard for Local and Metropolitan Area Networks and the Wireless Access Systems Part 16: Air Interface for Broadband*, IEEE Standard 802, 2009. [Online]. Available: http://standards.ieee.org/about/get/802/802.16.html

[2] R. D. Whitty, "Three-dimensional geolocation of mobile wimax subscribers," M.S. thesis, Naval Postgraduate School, Dec. 2010.

[3] Y. Zhao, "Standardization of mobile phone positioning for 3g systems," *IEEE Communications Magazine*, vol. 40, no. 7, pp. 108 –116, July 2002.

[4] S. S. Soliman and C. E. Wheatley, "Geolocation technologies and applications for third generation wireless," *Wireless Communications and Mobile Computing*, vol. 2, no. 3, pp. 229–251, 2002. [Online]. Available: http://dx.doi.org/10.1002/wcm.55

[5] A. Aloudat and K. Michael, "Toward the regulation of ubiquitous mobile government: a case study on location-based emergency services in Australia," *Electronic Commerce Research*, vol. 11, no. 1, pp. 31–74, 2011.

[6] F. Gustafsson and F. Gunnarsson, "Mobile positioning using wireless networks: possibilities and fundamental limitations based on available wireless network measurements," *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 41–53, Jul. 2005.

[7] K. Cheung, W. Ma, and H. So, "Accurate approximation algorithm for TOA-based maximum likelihood mobile location using semidefinite programming," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, vol. 2, May 2004, pp. 145–148.

[8] J. Caffery and G. Stuber, "Overview of radiolocation in CDMA cellular systems," *IEEE Communications Magazine*, vol. 36, no. 4, pp. 38–45, Apr. 1998.

[9] K. Cheung, H. So, W.-K. Ma, and Y. Chan, "Least squares algorithms for time-of-arrival-based mobile location," *IEEE Transactions on Signal Processing*, vol. 52, no. 4, pp. 1121–1130, Apr. 2004.

[10] K. Cheung and H. So, "A multidimensional scaling framework for mobile location using time-of-arrival measurements," *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 460–470, Feb. 2005.

[11] L. Cong and W. Zhuang, "Hybrid TDOA/AOA mobile user location for wideband CDMA cellular systems," *IEEE Transactions on Wireless Communications*, vol. 1, no. 3, pp. 439–447, Jul. 2002.

[12] Anonymous, "Follow me." *Economist*, vol. 394, no. 8672, p. 85, 2010.

[13] E. G. Bakhoum, “Closed-form solution of hyperbolic geolocation equations,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 4, pp. 1396–1404, Oct. 2006.

[14] J. Caffery Jr., “A new approach to the geometry of TOA location,” in *52nd IEEE Vehicular Technology Conference (IEEE VTS-Fall VTC)*, vol. 4, 2000, pp. 1943–1949.

[15] H. Koorapaty, H. Grubeck, and M. Cedervall, “Effect of biased measurement errors on accuracy of position location methods,” in *The Bridge to Global Integration. IEEE Global Telecommunications Conference, 1998 (GLOBECOM 98)*, vol. 3, 1998, pp. 1497–1502.

[16] D. Barber and J. McEachen, “Geolocation of WiMAX subscriber stations based on the timing adjust ranging parameter,” in *4th International Conference on Signal Processing and Communication Systems (ICSPCS)*, Dec. 2010, pp. 1–5.

[17] S. Drake and K. Dogancay, “Geolocation by time difference of arrival using hyperbolic asymptotes,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’04).*, vol. 2, no. 2, May 2004, pp. 361–364.

[18] D. Torrieri, “Statistical theory of passive location systems,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-20, no. 2, pp. 183–198, Mar. 1984.

[19] A. Roxin, J. Gaber, M. Wack, and A. Nait-Sidi-Moh, “Survey of wireless geolocation techniques,” in *IEEE Globecom Workshops*, Nov. 2007, pp. 1–9.

[20] T. Li, A. Ekpenyong, and Y.-F. Huang, “Source localization and tracking using distributed asynchronous sensors,” *IEEE Transactions on Signal Processing*, vol. 54, no. 10, pp. 3991–4003, Oct. 2006.

[21] B. Fang, “Simple solutions for hyperbolic and related position fixes,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 5, pp. 748–753, Sep. 1990.

[22] J. Abel, “A divide and conquer approach to least-squares estimation with application to range-difference-based localization,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP-89)*, vol. 4, May 1989, pp. 2144–2147.

[23] Y. Chan and K. Ho, “A simple and efficient estimator for hyperbolic location,” *IEEE Transactions onSignal Processing*, vol. 42, no. 8, pp. 1905–1915, Aug. 1994.

[24] K. Ho and Y. Chan, “Solution and performance analysis of geolocation by TDOA,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 4, pp. 1311–1322, Oct. 1993.

[25] K. Ho and Y. Chan, “Geolocation of a known altitude object from TDOA and FDOA measurements,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 3, pp. 770–783, Jul. 1997.

[26] K. Ho and W. Xu, "An accurate algebraic solution for moving source location using TDOA and FDOA measurements," *IEEE Transactions on Signal Processing*, vol. 52, no. 9, pp. 2453–2463, Sep. 2004.

[27] A. Bishop, B. Fidan, B. Anderson, K. Dogancay, and P. Pathirana, "Optimal range-difference-based localization considering geometrical constraints," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 3, pp. 289 –301, Jul. 2008.

[28] A. Urruela, J. Sala, and J. Riba, "Average performance analysis of circular and hyperbolic geolocation," *IEEE Transactions on Vehicular Technology*, vol. 55, no. 1, pp. 52–66, Jan. 2006.

[29] A. Amar and A. Weiss, "Localization of narrowband radio emitters based on Doppler frequency shifts," *IEEE Transactions on Signal Processing*, vol. 56, no. 11, pp. 5500–5508, Nov. 2008.

[30] M. Bshara, U. Orguner, F. Gustafsson, and L. Van Biesen, "Fingerprinting localization in wireless networks based on received-signal-strength measurements: A case study on WiMAX networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 1, pp. 283–294, Jan. 2010.

[31] K. Pahlavan, F. O. Akgul, M. Heidari, A. Hatami, J. M. Elwell, and R. D. Tingley, "Indoor geolocation in the absence of direct path," *IEEE Wireless Communications*, vol. 13, no. 6, pp. 50–58, Dec. 2006.

[32] Y. Qi, H. Kobayashi, and H. Suda, "Analysis of wireless geolocation in a non-line-of-sight environment," *IEEE Transactions on Wireless Communications*, vol. 5, no. 3, pp. 672–681, Mar. 2006.

[33] H. Miao, K. Yu, and M. Juntti, "Positioning for NLOS propagation: Algorithm derivations and Cramer-Rao bounds," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 5, pp. 2568–2580, Sep. 2007.

[34] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, 9th ed. Wiley, 2011, vol. 2.

[35] S. Laurila, *Electronic Surveying and Navigation*. Wiley, 1976.

[36] J. M. Rüeger, "Refractive index formulae for radio waves," in *Proceedings of the XXII FIG International Congress*. International Federation of Surveyors, Apr. 2002. [Online]. Available: http://www.fig.net/pub/fig_2002/Js28/JS28_rueger.pdf

[37] A. L. Buck, "New equations for computing vapor pressure and enhancement factor," *Journal of Applied Meteorology*, vol. 20, no. 12, pp. 1527–1532, 1981.

[38] B. Bean and G. Thayer, "Models of the atmospheric radio refractive index," *Proceedings of the IRE*, vol. 47, no. 5, pp. 740–755, May 1959.

[39] B. Bean, “The radio refractive index of air,” *Proceedings of the IRE*, vol. 50, no. 3, pp. 260–273, Mar. 1962.

[40] K. P. Birch and M. J. Downs, “An updated Edlén equation for the refractive index of air,” *Metrologia*, vol. 30, no. 3, p. 155, 1993. [Online]. Available: http://stacks.iop.org/0026-1394/30/i=3/a=004

[41] J. Boehm, R. Heinkelmann, P. J. Mendes Cerveira, A. Pany, and H. Schuh, “Atmospheric loading corrections at the observation level in VLBI analysis,” *Journal of Geodesy*, vol. 83, no. 11, pp. 1107–1113, 2009.

[42] K. P. Birch and M. J. Downs, “The results of a comparison between calculated and measured values of the refractive index of air,” *Journal of Physics E: Scientific Instruments*, vol. 21, no. 7, p. 694, 1988. [Online]. Available: http://stacks.iop.org/0022-3735/21/i=7/a=015

[43] S. Falodun and M. Ajewole, “Radio refractive index in the lowest 100-m layer of the troposphere in Akure, South Western Nigeria,” *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 68, no. 2, pp. 236–243, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364682605002907

[44] G. Chartrand, *Introductory Graph Theory*. Dover, 1985.

[45] *CRC Handbook of Standard Mathematical Tables and Formulae*, 29th ed., CRC, Boca Raton, FL, 1991.

[46] R. Miles and K. Hamilton, *Learning UML 2.0*, 1st ed. O’Reilly, Apr. 2006.

[47] M. A. Lombardi, “The use of GPS disciplined oscillators as primary frequency standards for calibration and metrology laboratories,” *NCLSI Measure J. Meas. Sci.*, vol. 3, no. 3, Sep. 2008.

[48] *IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology – Random Instabilities*, IEEE Standard 1139, 2008.

[49] *MATLAB Object-Oriented Programming*, R2012b ed., MathWorks, Natick, MA, 2012. [Online]. Available: www.mathworks.com/help/pdf_doc/matlab/matlab_oop.pdf

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California
3. Dr. Clark Robertson
   Naval Postgraduate School
   Monterey, CA
4. Dr. Murali Tummala
   Naval Postgraduate School
   Monterey, CA
5. Dr. John McEachen
   Naval Postgraduate School
   Monterey, CA
6. Jason McClintic
   Naval Postgraduate School
   Monterey, CA
7. CDR Owen Schoolsky, USN
   Navy Special Warfare Command
   N39/Force Tactical Information Operations Officer
   Coronado, CA